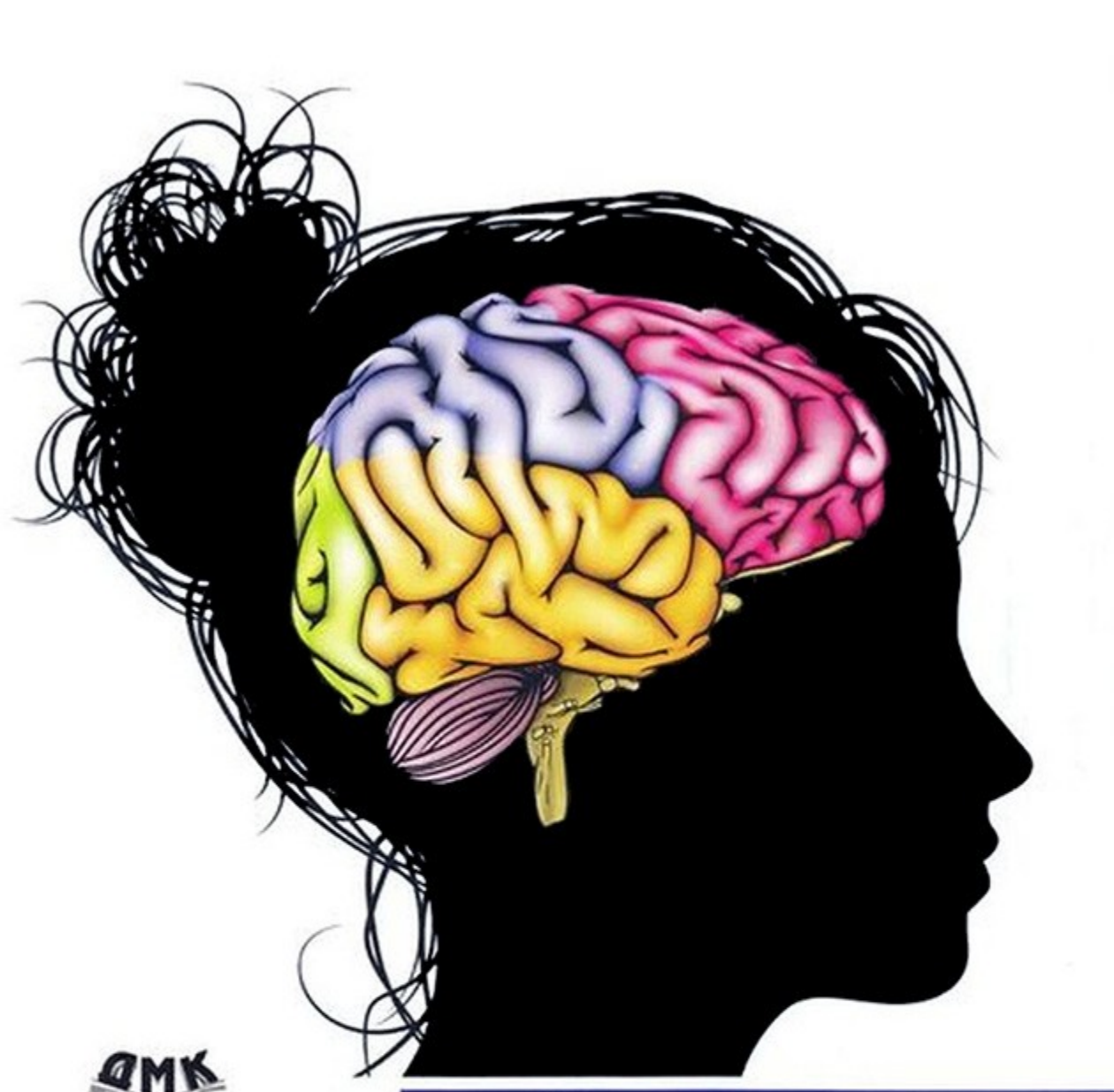


Йоав Гольдберг

Нейросетевые методы в обработке естественного языка

Yoav Goldberg
Bar Ilan University

Neural Network Methods for Natural Language Processing

Йоав Гольдберг
Университет имени Бар-Илана

Нейросетевые методы в обработке естественного языка

ДМК Пресс
издательство
Москва, 2019

УДК 004.89.032.26
ББК 32.813
Г63

Гольдберг Й.
Г63 Нейросетевые методы в обработке естественного языка / пер. с анг. А. А. Слинкина. – М.: ДМК Пресс, 2019. – 282 с.: ил.

ISBN 978-5-97060-754-1

Это классическое руководство посвящено применению нейросетевых моделей к обработке данных естественного языка (Natural Language Processing – NLP). Рассматриваются основы машинного обучения с учителем на лингвистических данных и применение векторных, а не символических представлений слов. Обсуждается абстракция графа вычислений, которая позволяет легко определять и обучать произвольные нейронные сети и лежит в основе современных программных нейросетевых библиотек. Также даются обзорные сведения специализированных нейросетевых архитектур, включая одномерные сверточные сети, рекуррентные нейронные сети, модели условной генерации и модели с механизмом внимания.

Издание предназначено студентам вузов, а также специалистам в области машинного перевода и нейронных сетей.

Предполагается знание теории вероятностей, алгебры и математического анализа, а также базовое владение алгоритмами и структурами данных.

УДК 004.89.032.26
ББК 32.813

ISBN 978-1-62705-298-6 (анг.)
ISBN 978-5-97060-754-1 (рус.)

Содержание

Об авторе

Йоав Гольдберг занимается обработкой естественного языка более десяти лет. Он работает старшим преподавателем на факультете информатики университета имени Бар-Илана в Израиле. До этого был исследователем в компании Google Research, Нью-Йорк. Получил степень доктора философии по информатике и обработке естественного языка в университете имени Бен-Гуриона в 2011 году. Регулярно участвует в мероприятиях по NLP и машинному обучению, является членом редколлегии журнала *Computational Linguistics*. Автор более 50 научных работ, получал премии за лучшую и выдающуюся работу на крупных конференциях по обработке естественного языка. В сферу научных интересов входят машинное обучение естественному языку, структурное предсказание, синтаксический анализ, обработка языков с развитым морфологическим строем, а в последние два года еще и нейросетевые модели с упором на рекуррентные нейронные сети.

Предисловие

Обработка естественного языка (Natural Language Processing – NLP) – термин, относящийся к различным способам вычислительной обработки человеческих языков. Сюда входят как алгоритмы, принимающие на входе созданный человеком текст, так и алгоритмы, порождающие тексты, которые выглядят как естественные. Потребность в таких алгоритмах постоянно растет: люди каждый код производят все возрастающие объемы текстов и ожидают, что компьютер будет общаться с ними на их родном языке. При этом задачи, возникающие при обработке естественного языка, очень сложны, поскольку человеческим языкам присуща неоднозначность, изменчивость, да и определены они нестрого.

Естественный язык по природе своей символьный, и первые попытки обработать его тоже были основаны на работе с символами: логике, правилах и онтологиях. Но крайняя неоднозначность и непостоянство языка потребовали алгоритмических подходов, в большей степени основанных на статистике. В настоящее время наиболее распространенные методы основаны на *статистическом машинном обучении*. Уже больше десяти лет в NLP преобладают подходы, основанные на линейных моделях обучения с учителем, такие как перцептроны, метод опорных векторов и логистическая регрессия. Модели обучаются на векторах очень высокой размерности, но при этом крайне разреженных.

Примерно в 2014 году в отрасли наметился переход от линейных моделей над разреженными входными данными к нелинейным нейросетевым моделям над плотными данными. Некоторые нейросетевые методы являются обобщениями линейных моделей и могут использоваться вместо них в линейных классификаторах. Другие, более продвинутые, требуют изменить взгляд на задачу, а взамен предлагают новые возможности моделирования. В частности, подходы, основанные на рекуррентных нейронных сетях (РНС), опираются на марковское предположение, характерное для последовательностных моделей, и позволяют иметь в качестве условий сколь угодно длинные последовательности, порождая при этом эффективные экстракторы признаков. Эти успехи привели к прорывам в языковом моделировании, автоматическом машинном переводе и других приложениях.

При всей своей эффективности нейросетевые методы по различным причинам задают высокий порог входа. В этой книге я сделал попытку познакомить как уже работающих в области NLP специалистов, так и новичков с предпосылками, терминологией, инструментами и методиками, необходимыми для понимания принципов, лежащих в основе нейросетевых моделей языка, чтобы они могли применять их в своей работе. Я также надеюсь познакомить специалистов по машинному обучению и нейронным сетям с предпосылками, терминологией, инструментами и образом мыслей, необходимыми для эффективного применения их знаний к работе с языковыми данными.

Наконец, льщу себя надеждой, что эта книга послужит сравнительно доступным (пусть и неполным) введением в NLP и машинное обучение для тех, кто только начинает изучение той или другой дисциплины.

Предполагаемая аудитория

Эта книга рассчитана на читателей, имеющих техническую подготовку в области информатики или смежных с ней областях знаний и желающих поскорее освоить нейросетевые методы обработки естественного языка. Хотя предполагается, что основными читателями книги будут студенты магистратуры, специализирующиеся на обработке лингвистической информации и машинном обучении, я старался сделать книгу полезной и для уже сложившихся исследователей в области NLP или машинного обучения (путем включения дополнительных материалов), а также для тех, у кого еще нет опыта работы с машинным обучением или NLP (благодаря построению изложения с основ данной темы). Но последней группе читателей, очевидно, придется работать усерднее.

Хотя книга самодостаточна, я все же предполагаю у читателей знание математики, в частности теории вероятностей, алгебры и математического анализа в объеме младших курсов университета, а также базовое владение алгоритмами и структурами данных. Знакомство с машинным обучением было бы очень полезно, но не обязательно.

Эта книга родилась из обзорной статьи [Goldberg, 2016], существенно расширенной и несколько реорганизованной с целью сделать ее полнее и осветить некоторые темы, по разным причинам не вошедшие в обзор. В книге также имеется гораздо больше конкретных примеров приложения нейронных сетей к лингвистическим данным, чем в обзоре. Кроме того, книга рассчитана на читателей, не имеющих подготовки в области NLP или машинного обучения, тогда как в обзоре предполагается наличие соответствующих знаний. Вообще, читатели, знакомые с обработкой естественного языка в том виде, в каком она практиковалась между 2006 и 2014 годом, т. е. существенно опирающейся на машинное обучение и линейные модели, вероятно, сочтут, что журнальный вариант короче и лучше организован. Но и такие читатели могут извлечь пользу из глав о погружении слов (10 и 11), из главы об условной генерации с помощью PHC (17) и из глав, посвященных структурному предсказанию и многозадачному обучению (19 и 20).

Задачи этой книги

Книга задумана самодостаточной, представляющей различные подходы в рамках единой системы обозначений и методики. Однако основное ее назначение – служить введением в механизм нейронных сетей (глубокого обучения) и их применение к лингвистическим данным, а не углубленное изложение основ теории машинного обучения и технологии обработки естественного языка. Отсылаю читателя к другим ресурсам, если такая необходимость возникнет.

Аналогично книга не является исчерпывающим источником для тех, кто намерен развивать нейросетевые подходы (хотя и может служить неплохой отправной точкой). Скорее, она рассчитана на читателей, интересующихся современным состоянием технологии и поиском полезных и нестандартных путей ее применения к стоящим перед ними задачам обработки языка.

Для дальнейшего чтения. Общее всестороннее обсуждение нейронных сетей, стоящей за ними теории, продвинутых методов оптимизации и других тем следует искать в других источниках. В особенности рекомендую книгу Bengio et al. [2016][1].

Доступное, хотя и строгое введение в практическое машинное обучение имеется в бесплатной публикации Daumé III [2015], которую я горячо рекомендую. Теоретические вопросы машинного обучения прекрасно освещены в бесплатной книге Shalev-Shwartz and Ben-David [2014][2] и в учебнике Mohri et al. [2012].

Строгое введение в NLP см. в книге Jurafsky and Martin [2008]. В книге по информационному поиску Manning et al. [2008] также имеется информация, относящаяся к работе с языковыми данными.

Наконец, желающим получить базовые лингвистические знания рекомендуем книгу Bender [2013], вышедшую в этой же серии. Она содержит лаконичное, но достаточно полное изложение вопроса, ориентированное на любителей вычислительных методов. Стоит также познакомиться с первыми главами вводного учебника Sag et al. [2003] по грамматике языков.

На момент написания этой книги исследования в области нейронных сетей и глубокого обучения идут полным ходом. Сказать, что такое текущее состояние дел, невозможно, и я не могу надеяться на сохранение актуальности. Поэтому моя задача – охватить в основном уже устоявшиеся, надежные методы, доказавшие свою работоспособность в нескольких приложениях, а также несколько избранных методов, которые еще нельзя считать полноценными, но которые я считаю либо вошедшими в практику, либо многообещающими.

Йоав Гольдберг
Март 2017

[1] *Иошуа Бенджио, Ян Гудфеллоу, Аарон Курвилль*. Глубокое обучение. М.: ДМК Пресс, 2017.

[2] *Шай Шалев-Шварц, Шай Бен-Давид.* Идеи машинного обучения. М.: ДМК Пресс, 2019.

Благодарности

Эта книга выросла из моей обзорной статьи [Goldberg, 2016], которая, в свою очередь, появилась вследствие моего недовольства отсутствием четко организованного и ясно изложенного материала о пересечении глубокого обучения с обработкой естественного языка – темы, которую я тогда пытался изучить сам и преподавать своим студентам и коллегам. Поэтому я в долгу перед многими людьми, поделившимися своими замечаниями об обзоре (в различных формах – от замечаний по начальным редакциям до комментариев после публикации), а также теми, кто следил за ходом работы над рукописью книги. Кто-то делился замечаниями лично, кто-то – по электронной почте, а кто-то – в случайных беседах в Твиттере. На книгу также оказали влияние люди, которые не комментировали ее (а иногда даже и не читали), но обсуждали со мной затронутые в ней темы. Одни из них – специалисты по глубокому обучению, другие – по NLP, третьи – по тому и другому, а четвертые пытались изучать эти дисциплины. Кто-то (немногие) присылал очень подробные комментарии, другие обсуждали мелкие детали, а кто-то занимает промежуточную позицию. Но все они повлияли на окончательный текст книги. Перечисляю их в алфавитном порядке: Йоав Арци (Yoav Artzi), Йонатан Ауманн (Yonatan Aumann), Джейсон Бэлбридж (Jason Baldridge), Мигель Баллестерос (Miguel Ballesteros), Мохит Бансал (Mohit Bansal), Марко Барони (Marco Baroni), Тал Баумель (Tal Baumel), Сэм Боумэн (Sam Bowman), Джордан Бойд-Грабер (Jordan Boyd-Graber), Крис Брокетт (Chris Brockett), Минь Вэй Чанг (Ming-Wei Chang), Дэвид Чианг (David Chiang), Куюн Хуэн Чо (Kyunghyun Cho), Гжегож Хрупала (Grzegorz Chrupala), Александр Кларк (Alexander Clark), Рафаэль Коэн (Raphael Cohen), Райан Коттерелл (Ryan Cotterell), Хэл Дауме III (Hal Daumé III), Николас Дронен (Nicholas Dronen), Крис Дайер (Chris Dyer), Якоб Эйзенштейн (Jacob Eisenstein), Джейсон Эйзнер (Jason Eisner), Майкл Эльхадад (Michael Elhadad), Йад Фаэк (Yad Faeq), Манаал Фаруки (Manaal Faruqui), Амир Глоберсон (Amir Globerson), Фредерик Годэн (Fréderic Godin), Эдвард Грефенштетте (Edward Grefenstette), Мэтью Хоннибал (Matthew Honnibal), Дирк Хови (Dirk Hovy), Моше Коппель (Moshe Koppel), Анджелики Лазариду (Angeliki Lazaridou), Тал Линзен (Tal Linzen), Тханг Луонг (Thang Luong), Крис Мэннинг (Chris Manning), Стивен Мерити (Stephen Merity), Пал Мишель (Paul Michel), Маргарет Митчелл (Margaret Mitchell), Пьеро Молино (Piero Molino), Грэм Нойбиг (Graham Neubig), Иоаким Нивре (Joakim Nivre), Брендан О'Коннор (Brendan O'Connor), Никос Паппас (Nikos Pappas), Фернандо Перейра (Fernando Pereira), Барбара Планк (Barbara Plank), Ана-Мария Попеску (Ana-Maria Popescu), Делип Рао (Delip Rao), Тим Роктэшел (Tim Rocktäschel), Дэн Рот (Dan Roth), Александр Раш (Alexander Rush), Наоми Сафра (Naomi Saphra), Джаме Седдах (Djamé Seddah), Эрел Сегал-Халеви (Erel Segal-Halevi), Ави Шмидман (Avi Shmidman), Шалтиэль Шмидман (Shaltiel Shmidman), Ноа Смит (Noah Smith), Андерс Сёго (Anders Søgaard), Абе Стэнвей (Abe Stanway), Эмма Штрубелль (Emma Strubell), Сандип Субраманиан (Sandeep Subramanian), Ли Лин Тань (Liling Tan), Реут Царфати (Reut Tsarfaty), Питер Тэрни (Peter Turney), Тим Виэйра (Tim Vieira), Ориол Виньялс (Oriol Vinyals), Андреас Влачос (Andreas Vlachos), Вэнь Пэн Инь (Wenpeng Yin), Торстен Цэш (Torsten Zesch).

Разумеется, в этот список не вошли очень многие исследователи, с которыми я общался благодаря их публикациям в академических изданиях.

Книга также много приобрела – и даже сформировалась – в результате моего взаимодействия с группой обработки естественного языка в университете имени Бар-Илана (и ее ответвлениях): Йосси Ади (Yossi Adi), Рое Ахарони (Roee Aharoni), Одед Аврахам (Oded Avraham), Идо Даган (Ido Dagan), Джессика Фиклер (Jessica Ficler), Якоб Голдбергер (Jacob Goldberger), Хила Гонен (Hila Gonen), Джозеф Кешет (Joseph Keshet), Элияху Кипервассер (Eliyahu Kiperwasser), Рон Конигсберг (Ron Konigsberg), Омер Леви (Omer Levy), Орен Меламуд (Oren Melamud), Габриэль Становски (Gabriel Stanovsky), Ори Шапира (Ori Shapira), Миха Шлэйн (Micah Shlain), Веред Шварц (Vered Shwartz), Хиллэл Тауд-Табиб (Hillel Taub-Tabib), Рэйчел Уитис (Rachel Wities). Большинство их них входит в оба списка, но я стремился к краткости.

Анонимные рецензенты книги и обзорной статьи – хотя они и не названы (а иногда сильно раздражали) – внесли немало замечаний, предложений и исправлений, которые, безусловно, значительно улучшили окончательную редакцию во многих отношениях. Спасибо всем вам! И отдельная благодарность Грэму Хирсту (Graeme Hirst), Майклу Моргану (Michael Morgan), Саманте Дрейпер (Samantha Draper) и К. Л. Тондо (C.L. Tondo), которые координировали всю работу.

Как обычно, все ошибки – целиком и полностью моя вина. Сообщите мне, если что-то найдете, чтобы внести исправления в следующее издание, если таковое когда-нибудь состоится.

Наконец, хочу поблагодарить свою жену Ноа, которая терпела и поддерживала меня во время писательских загулов, своих родителей Эстер и Авнера и брата Надав, которые зачастую переживали из-за моей идеи написать книгу больше, чем я сам. А также персонал кафе The Streets и кафе Shne'or, которые кормили и поили меня на протяжении всего процесса написания, почти не отвлекая от дела.

Йоав Гольдберг
март 2017

Предисловие от издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте **www.dmkpress.com**, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу **dmkpress@gmail.com**, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу **http://dmkpress.com/authors/publish_book/** или напишите в издательство по адресу **dmkpress@gmail.com**.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг — возможно, ошибку в тексте или в коде, — мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу **dmkpress@gmail.com**, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Packt очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или вебсайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу электронной почты **dmkpress@gmail.com** со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Глава 1

Введение

1.1. Проблемы, стоящие перед обработкой естественного языка

Обработка естественного языка – это наука о проектировании методов и алгоритмов, которые принимают или порождают неструктурированные данные естественного языка. Человеческие языки в высшей степени неоднозначны (сравните фразы «Я ел пиццу с друзьями» и «Я ел пиццу с оливками») и вариативны (смысл фразы «Я ел пиццу с друзьями» можно выразить и так: «Мы с друзьями разделили пиццу»). Языки также постоянно изменяются и развиваются. Люди прекрасно справляются с порождением и пониманием языковых конструкций, они способны выразить, воспринять и интерпретировать фразы с утонченным смыслом, изобилующим нюансами. Но, будучи замечательными *пользователями* языка, люди очень плохо справляются с формальным описанием правил, *управляющих* языком.

Поэтому понимание и порождение языка с помощью компьютеров – чрезвычайно трудная задача. Лучшие методы работы с языковыми данными основаны на алгоритмах машинного обучения с учителем, которые пытаются вывести паттерны и закономерности использования из множества предварительно аннотированных пар входных и выходных текстов. Рассмотрим, к примеру, задачу классификации документа по одной из четырех категорий: `Спорт`, `Политика`, `Светская хроника` и `Экономика`. Очевидно, что содержащиеся в документе слова дают вполне определенные указания, но какое именно указание дает данное слово? Выписать соответствующие правила довольно трудно. Однако читатели легко могут отнести документ к теме, а затем, основываясь на нескольких сотнях аннотированных таким образом документов в каждой категории, алгоритм машинного обучения с учителем может вывести паттерны использования слов, которые помогут классифицировать новые документы. Методы машинного обучения отлично работают в задачах, где определить хороший набор правил очень трудно, а аннотировать входные примеры выходными метками сравнительно просто.

Помимо проблем, связанных с обработкой неоднозначных и вариативных входных данных в системе с плохо определенными и отсутствующими наборами правил, у естественного языка есть и дополнительные свойства, которые еще больше затрудняют разработку вычислительных подходов на основе машинного обучения: *дискретность*, *композиционность* и *разреженность*.

Язык по природе своей символический и дискретный. Основными элементами письменного языка являются литеры. Литеры образуют слова, обозначающие предметы, понятия, события, действия и идеи. Литеры и слова – дискретные символы: слова «гамбургер» или «пицца» вызывают определенные мысленные образы, но также являются разными символами, смысл которых отделен от них и интерпретируется в нашем мозгу. Не существует внутренней связи между «гамбургером» и «пиццей», которую можно было бы вывести из самих символов или составляющих их букв. Сравните это с таким понятием, как цвет, играющий важнейшую роль в машинном зрении, или звуковые сигналы: эти величины непрерывны, что позволяет, например, перейти от цветного изображения к тоновому с помощью простой математической операции, или сравнить два цвета на основе внутренне присущих им свойств, например оттенка и яркости. Со словами так не получится – не существует простой операции, которая позволила бы перейти от слова «красный» к слову «розовый» без использования большой справочной таблицы или словаря.

Язык также обладает свойством композиционности: буквы образуют слова, слова образуют фразы и предложения. Смысл фразы может быть больше смысла составляющих ее слов и определяется набором запутанных правил. Поэтому, чтобы интерпретировать текст, приходится подняться над уровнем букв и слов и рассматривать длинные последовательности слов, например предложения или даже полные документы.

Сочетание описанных выше свойств ведет к *разреженности данных*. Число комбинаций слов (дискретных символов), имеющих смысл, практически бесконечно. Число допустимых предложений огромно, нет никакой надежды перечислить их все. Откройте любую книгу – подавляющее большинство встречающихся в ней предложений вы никогда раньше не видели и не слышали. Более того, вполне вероятно, что многие последовательности четырех слов, встречающиеся в этой книге, тоже не попадались вам ранее. Если вы возьмете газету, вышедшую каких-то 10 лет назад, или попробуете вообразить газету, которая выйдет через 10 лет, то многие слова, особенно имена людей, названия торговых марок и компаний, а также жаргонные словечки и технические термины, покажутся незнакомыми. Нет никакого очевидного способа получить одно предложение из другого путем обобщения или определить сходство предложений, не зависящее от их смысла, который для нас является ненаблюдаемой величиной. Поэтому задача обучения на примерах становится крайне трудной: даже располагая гигантским набором примеров, мы все равно с большой вероятностью будем наблюдать события, которые в этом наборе не встречались и сильно отличаются от всех встречающихся примеров.

1.2. Нейронные сети и глубокое обучение

Глубокое обучение – раздел машинного обучения. Это другое название нейронных сетей – семейства методов обучения, исторически появившегося в результате изучения способов вычислений в мозге. Его еще можно охарактеризовать как обучение параметрических дифференцируемых функций[1]. Само название «глу-

[1] В этой книге превалирующим является математический, а не основанный на аналогии с мозгом взгляд.

бокое обучение» связано с тем, что многочисленные слои, образованные этими дифференцируемыми функциями, зачастую сцеплены друг с другом.

Хотя машинное обучение в целом можно описать как обучение делать предсказания на основе прошлых наблюдений, в глубоком обучении упор делается на том, чтобы научиться не только делать прогнозы, но и *правильно представлять* данные, чтобы они были пригодны для предсказания. Располагая большим набором соответствий между входами и выходами, мы подаем данные на вход сети, которая последовательно преобразует их, пока на выходе не получится готовое предсказание. Какие преобразования выполнять, сеть обучается на основе соответствий между входами и выходами, стремясь выбирать каждое преобразование так, чтобы было проще соотнести данные с желаемой меткой.

Проектировщик-человек отвечает за разработку архитектуры сети и режима обучения, обеспечение сети надлежащим набором обучающих примеров и выбор подходящего способа кодирования входных данных, а трудную работу по обучению правильному представлению берет на себя сеть при поддержке со стороны сетевой архитектуры.

1.3. Глубокое обучение в NLP

Нейронные сети дают эффективный механизм обучения, чрезвычайно привлекательный для использования в задачах обработки естественного языка. Главный компонент языковой нейронной сети – *слой погружения*, т. е. отображение дискретных символов на непрерывные векторы в пространстве сравнительно небольшой размерности. В результате погружения слова преобразуются из изолированных дискретных символов в математические объекты, над которыми можно производить различные действия. В частности, если за меру расстояния между словами взять расстояние между векторами, то будет проще обобщить влияние одного слова на другое. Такое представление слов векторами сеть находит в процессе обучения. Поднимаясь вверх по иерархии, сеть также обучается комбинировать векторы слов способами, полезными для предсказания. Эта возможность до некоторой степени компенсирует дискретность и разреженность данных.

Существует два основных вида архитектуры нейронных сетей, которые можно по-разному комбинировать: сети прямого распространения и рекуррентные/рекурсивные сети.

Сети прямого распространения, в частности многослойные перцептроны (МСП), позволяют работать с входными данными фиксированного размера или с данными переменного размера, если можно не обращать внимания на порядок элементов. Если загрузить в сеть множество входных компонентов, то она обучится комбинировать их осмысленными способами. МСП можно использовать в тех случаях, где раньше применялась линейная модель. Нелинейность сети, а также возможность интегрировать в нее ранее обученные погружения слов часто приводят к выдающейся точности классификации.

Сверточные нейронные сети – это специализированные архитектуры, отличающиеся способностью выделять локальные паттерны в данных: на вход им подаются данные произвольного размера, а они выделяют осмысленные локальные паттерны, чувствительные к порядку слов, независимо от того, в каком месте входных данных они встречаются. Они очень хорошо справляются с идентифи-

кацией фраз в изъявительном наклонении и идиом заранее ограниченной длины в длинных предложениях или документах.

Рекуррентные нейронные сети (РНС) – это специализированные модели для последовательных данных. Они принимают входную последовательность объектов и порождают вектор фиксированной длины, который подытоживает ее. Смысл слов «подытожить последовательность» зависит от задачи (например, информация, необходимая для ответа на вопрос об эмоциональной окраске предложения, отличается от той, что необходима для ответа на вопрос о его грамматической правильности). Поэтому рекуррентные сети редко используются сами по себе, а их ценность заключается в том, что это допускающие обучение компоненты, которые можно подать на вход другим компонентам сети и обучить совместной работе. Например, выход рекуррентной сети можно подать на вход сети прямого распространения, которая попытается предсказать некоторое значение. Рекуррентные сети являются весьма выразительными моделями для последовательностей и являют собой, пожалуй, самое полезное, что могут предложить нейронные сети обработке языков. Они позволяют отказаться от *марковского предположения*, преобладавшего в NLP в течение нескольких десятилетий, и проектировать модели, в которых условиями могут быть целые предложения. При этом они могут при необходимости учитывать порядок слов и не подвержены проблемам статистического оценивания, проистекающим из разреженности данных. Эта возможность дает заметный выигрыш в *языковом моделировании* – задаче о предсказании вероятности следующего слова в последовательности (или, что то же самое, вероятности предложения), – которое является краеугольным камнем многих приложений NLP. Рекурсивные сети обобщают рекуррентные сети с последовательностей на деревья.

Многие задачи в естественном языке *структурированы*, т. е. нуждаются в порождении сложных выходных структур типа последовательностей или деревьев. Нейросетевые модели пригодны и для этих целей – либо путем адаптации известных алгоритмов структурного предсказания для линейных моделей, либо благодаря использованию новых архитектур, таких как модели последовательность-в-последовательность (кодировщик–декодер), которые мы будем называть в этой книге *моделями условной генерации* (conditioned generation model).

Наконец, многие языковые задачи предсказания связаны друг с другом в том смысле, что знание о том, как решить одну из них, помогает при обучении другим. Кроме того, если *аннотированных учителем* обучающих примеров может не хватать, то уж недостатка в исходных (неаннотированных) текстовых данных точно не наблюдается. Можно ли обучаться на основе неаннотированных данных или результатов обучения родственных задач? Нейронные сети предлагают весьма интересные возможности как для многозадачного обучения (multitask learning – MTL), т. е. обучения на основе результатов для родственных задач, так и для обучения с частичным привлечением учителя (обучения на внешних неаннотированных данных).

1.3.1. Истории успеха

Полносвязные нейронные сети прямого распространения (МСП) в большинстве случаев могут использоваться всюду, где обычно применяется линейный обучаемый. Сюда относятся бинарные и многоклассовые проблемы обучения, а также

более сложные структурные проблемы предсказания. Нелинейность сети, а равно возможность простой интеграции с предобученными погружениями слов, часто ведет к выдающейся верности классификации. В серии работ[1] удалось добиться улучшения результатов синтаксического анализа, всего лишь заменив линейную модель анализатора полносвязной сетью прямого распространения. Прямолинейное применение сети прямого распространения в качестве замены классификатора (обычно в сочетании с использованием предобученных векторов слов) дает преимущества во многих лингвистических задачах, в том числе: основополагающей задаче языкового моделирования[2], суперразметке с комбинаторными категориальными грамматиками (CCG)[3], прослеживании состояния диалога[4] и предупорядочении для статистического машинного перевода[5]. В работе Iyyer et al. [2015] продемонстрировано, что многоуровневые сети прямого распространения могут давать конкурентоспособные результаты в задачах классификации по эмоциональной окраске и для ответов на вопросы на основе опубликованных сведений о фактах. В работах Zhou et al. [2015] и Andor et al. [2016] такие сети интегрированы в системы структурного предсказания на основе лучевого поиска, что позволило добиться фантастической верности в задачах синтаксического анализа, разметки последовательностей и других.

Сети со сверточными и пулинговыми слоями полезны для задач классификации, в которых мы ожидаем найти сильные локальные признаки, касающиеся принадлежности к классам, но эти признаки могут встречаться в разных местах входных данных. Например, в задаче классификации документов одна ключевая фраза (или *n*-грамма) может помочь в определении темы документа [Johnson and Zhang, 2015]. Мы хотели бы обучить систему таким последовательностям слов, которые служат хорошими тематическими индикаторами, не думая о том, в каком месте документа они встречаются. Сверточные и пулинговые слои позволяют моделировать обучение поиску таких локальных позиционно-независимых индикаторов. Сверточно-пулинговая архитектура продемонстрировала многообещающие результаты на многих задачах, включая классификацию документов[6], категоризацию коротких текстов[7], классификацию по эмоциональной окраске[8], классификацию по типу связи между сущностями[9], обнаружение событий[10], обнаружение перефразирования[11], пометку семантических ролей[12], ответы на вопросы[13], прогнозирование кассового сбора от проката фильмов на основе критических от-

---

1 [Chen and Manning, 2014, Durrett and Klein, 2015, Pei et al., 2015, Weiss et al., 2015].
2 *См*. главу 9, а также Bengio et al. [2003], Vaswani et al. [2013].
3 [Lewis and Steedman, 2014].
4 [Henderson et al., 2013].
5 [de Gispert et al., 2015].
6 [Johnson and Zhang, 2015].
7 [Wang et al., 2015a].
8 [Kalchbrenner et al., 2014, Kim, 2014].
9 [dos Santos et al., 2015, Zeng et al., 2014].
10 [Chen et al., 2015, Nguyen and Grishman, 2015].
11 [Yin and Schütze, 2015].
12 [Collobert et al., 2011].
13 [Dong et al., 2015].

зывов[1], моделирование интересности текста[2] и моделирование связи между последовательностями литер и метками частей речи[3].

В естественном языке часто приходится работать со структурированными данными произвольного размера, например последовательностями и деревьями. Мы хотели бы уметь улавливать закономерности в таких структурах или моделировать сходство между ними. Рекуррентные и рекурсивные архитектуры позволяют работать с последовательностями и деревьями, сохраняя при этом большой объем структурной информации. Рекуррентные сети [Elman, 1990] проектировались для моделирования последовательностей, а рекурсивные [Goller and Küchler, 1996] являются обобщениями рекуррентных сетей для обработки деревьев. Показано, что рекуррентные модели дают прекрасные результаты в языковом моделировании[4], а также в разметке последовательностей[5], машинном переводе[6], грамматическом разборе[7] и многих других задачах, включая нормализацию зашумленного текста[8], прослеживание состояния диалога[9], генерацию ответов[10] и моделирование связей между последовательностями литер и метками частей речи[11].

Показано, что рекурсивные модели дают лучшие в отрасли или близкие к лучшим результаты в задачах переранжирования результатов синтаксического анализа на основе грамматики составляющих[12] и зависимостей[13], анализа повествовательной линии[14], классификации семантических связей[15], распознавания политической идеологии на основе деревьев разбора[16], классификации по эмоциональной окраске[17], классификации по эмоциональной окраске в зависимости от цели[18] и ответов на вопросы[19].

1.4. Состав и организация материала

Книга состоит из четырех частей. Часть I представляет собой введение в основные механизмы обучения, которые будут использоваться в книге: обучение с учите-

---

1 [Bitvai and Cohn, 2015].
2 [Gao et al., 2014].
3 [dos Santos and Zadrozny, 2014].
4 Перечислим несколько заметных работ: Adel et al. [2013], Auli and Gao [2014], Auli et al. [2013], Duh et al. [2013], Jozefowicz et al. [2016], Mikolov [2012], Mikolov et al. [2010, 2011].
5 [Irsoy and Cardie, 2014, Ling et al., 2015b, Xu et al., 2015].
6 [Cho et al., 2014b, Sundermeyer et al., 2014, Sutskever et al., 2014, Tamura et al., 2014].
7 [Dyer et al., 2015, Kiperwasser and Goldberg, 2016b, Watanabe and Sumita, 2015].
8 [Chrupala, 2014].
9 [Mrkšić et al., 2015].
10 [Kannan et al., 2016, Sordoni et al., 2015].
11 [Ling et al., 2015b].
12 [Socher et al., 2013a]
13 [Le and Zuidema, 2014, Zhu et al., 2015a].
14 [Li et al., 2014].
15 [Hashimoto et al., 2013, Liu et al., 2015].
16 [Iyyer et al., 2014b].
17 [Hermann and Blunsom, 2013, Socher et al., 2013b].
18 [Dong et al., 2014].
19 [Iyyer et al., 2014a].

лем, МСП, градиентные методы обучения и абстракция графа вычислений для реализации и обучения нейронных сетей. В части II механизмы, описанные в первой части, соединяются с лингвистическими данными и объясняется, как их интегрировать с нейронными сетями. Здесь же обсуждаются алгоритмы погружения слов и дистрибутивная гипотеза, а также подходы к языковому моделированию на основе сетей прямого распространения. В части III речь пойдет о специализированных архитектурах и РНС для моделирования последовательностей и стеков. РНС – основное новшество, разработанное для применения нейронных сетей к лингвистическим данным, часть III посвящена по преимуществу именно им, включая обеспечиваемую ими инфраструктуру условной генерации и модели внимания. Часть IV – собрание различных дополнительных тем: применение рекурсивных сетей для моделирования деревьев, модели структурного предсказания и многозадачное обучение.

Часть I, посвященная основам нейронных сетей, состоит из четырех глав. В главе 2 вводятся основные понятия машинного обучения с учителем, параметрические функции, линейные и логлинейные модели, регуляризация и функции потерь, обучение как оптимизация и градиентные методы обучения. Материал излагается с азов и необходим для чтения последующих глав. Читатели, знакомые с основами теории обучения и градиентным обучением, могут пропустить эту главу. В главе 3 описывается главное ограничение нейронных сетей прямого распространения и МСП, обсуждается определение многослойных сетей, их теоретическая ценность и основные составные части, например нелинейность и функции потерь. В главе 4 вводятся нейронные сети прямого распространения и МСП. Обсуждается определение многослойных сетей, их теоретическая ценность и такие стандартные понятия, как нелинейность и функции потерь. Глава 5 посвящена обучению нейронных сетей. Вводится абстракция графа вычислений, которая позволяет автоматически вычислять градиенты в произвольных сетях (алгоритм обратного распространения), и приводится несколько важных советов и приемов эффективного обучения сети.

Часть II, посвященная лингвистическим данным, состоит из семи глав. В главе 6 описана типология проблем лингвистической обработки и обсуждаются источники информации (признаков), доступные нам при работе с лингвистическими данными. В главе 7 приводятся конкретные примеры, показывающие, как признаки, описанные в предыдущей главе, используются для решения различных задач обработки естественного языка. Читатели, знакомые с лингвистической обработкой, могут пропустить эти две главы. В главе 8 материал глав 6 и 7 соединяется с нейронными сетями и обсуждаются различные способы кодирования лингвистических признаков для подачи на вход нейронной сети. В главе 9 мы познакомимся с задачей языкового моделирования и архитектурой нейросетевой модели языка с прямым распространением. Тем самым будет проложен путь к обсуждению предобученных погружений слов в следующих главах. Глава 10 посвящена распределенным и дистрибутивным подходам к представлению значений слов. Вводится подход к дистрибутивной семантике на основе матрицы слово–контекст, а также алгоритмы погружения слов, вдохновленные нейросетевым языковым моделированием, в частности `GloVe` и `Word2Vec`; обсуждается их связь с дистрибутивными методами. В главе 11 речь пойдет об использовании погружения слов вне контекста нейронных сетей. Наконец, в главе 12 представлен пример сети прямого

распространения, ориентированной конкретно на задачу логического вывода на естественном языке.

Часть III, где вводятся специализированные сверточные и рекуррентные архитектуры, насчитывает пять глав. В главе 13 рассматриваются сверточные сети, специализированные для обучения информативным *n*-граммным паттернам. Обсуждается также альтернативный метод хеш-ядра. Оставшиеся в этой части главы 14–17 посвящены РНС. В главе 14 описывается абстракция РНС для моделирования последовательностей и стеков. В главе 15 рассматриваются конкретные примеры РНС, в том числе простая РНС (известная также как РНС Элмана) и вентильные архитектуры, в том числе долгая краткосрочная память (LSTM) и вентильный рекуррентный блок (GRU). В главе 16 приведены примеры моделирования с помощью абстракции РНС в конкретных приложениях. Наконец, в главе 17 описывается инфраструктура условной генерации – основного метода моделирования в современных системах машинного перевода, – а также моделирование предложений без учителя и многие другие инновационные приложения.

Часть IV представляет собой собрание продвинутых и немагистральных тем и состоит из трех глав. В главе 18 вводится понятие древовидных рекурсивных сетей для моделирования деревьев. Хотя это семейство моделей выглядит весьма привлекательно, оно пока находится в стадии исследований, и убедительные истории успеха – дело будущего. Тем не менее о нем должны знать ученые, собирающиеся развивать технику моделирования и дальше. Читатели, которых в основном интересуют зрелые, доказавшие свою надежность методы, могут пропустить эту главу. Глава 19 посвящена структурному предсказанию. Она носит технический характер. Читатели, специально интересующиеся этой темой или уже знакомые с методами структурного предсказания для линейных моделей или для обработки языка, вероятно, оценят приведенный здесь материал. Остальные могут спокойно пропустить эту главу. Наконец, в главе 20 представлено многозадачное обучение и обучение с частичным привлечением учителя. Нейронные сети предлагают широкие возможности для того и другого. Это важные методы, которые пока находятся в стадии исследований. Но уже существующие приемы реализовать относительно просто, и они приносят реальный выигрыш. С технической точки зрения, глава не особенно сложная, поэтому рекомендуется всем читателям.

ЗАВИСИМОСТИ Большинство глав зависят от предшествующих им. Исключение составляют первые две главы части II, которые не зависят от предыдущих и могут читаться в любом порядке. Некоторые главы и разделы можно пропустить без ущерба для понимания остального материала. К ним относятся раздел 10.4 и глава 11, в которых речь идет о деталях алгоритмов погружения слов и использовании погружений вне контекста нейронных сетей; глава 12, в которой описывается специальная архитектура для работы с набором данных Stanford Natural Language Inference (SNLI), ориентированным на естественно-языковой вывод; глава 13, где описываются сверточные сети. При чтении материала о рекуррентных сетях можно без особого ущерба пропустить главу 15, содержащую подробные сведения о конкретных архитектурах. Главы в части IV по большей части не зависят друг от друга, их можно читать в любом порядке или пропустить вовсе.

1.5. Что не вошло в книгу

Книга посвящена применениям нейронных сетей к задачам обработки языка. Но некоторые разделы лингвистической обработки с помощью нейронных сетей сознательно опущены. Особенно я уделял внимание обработке письменного языка, а речевые данные и звуковые сигналы не получили освещения. В рамках письменного языка я рассматриваю низкоуровневые, относительно корректно поставленные задачи, не вторгаясь в такие области, как диалоговые системы, реферирование документов или вопросно-ответные системы, которые считаю открытыми проблемами. Хотя применение описанных методов, возможно, позволит добиться прогресса в решении этих задач, я не привожу никаких примеров и не вдаюсь в их обсуждение. Точно так же за рамками книги остался семантический анализ. Мультимодальные приложения, связывающие лингвистические данные с другими модальностями, например техническим зрением или базами данных, лишь кратко упомянуты. Наконец, обсуждаются в основном англоязычные тексты, а языки с более развитым морфологическим строем и недостаточным объемом аннотированных ресурсов почти не затрагиваются.

Не рассматриваются также некоторые базовые вопросы. В частности, для качественной лингвистической обработки критически важны *надлежащая оценка* и *аннотирование данных*. Оба вопроса оставлены за рамками книги, но читатель должен знать об их существовании.

Надлежащая оценка включает выбор подходящих метрик для оценивания качества алгоритма на данной задаче, передовые практики, честное сравнение с другими работами, выполнение анализа ошибок и оценку статистической значимости.

Аннотирование данных – это хлеб насущный систем NLP. Без данных мы не сможем обучить модели с учителем. В исследованиях мы очень часто используем «стандартные» аннотированные данные, созданные кем-то еще. Но все равно важно знать об источнике этих данных и принимать во внимание следствия, вытекающие из процесса создания набора. Аннотирование данных – весьма обширная тема, включающая корректную постановку задачи, разработку инструкций по аннотированию, принятие решения о выборе источника аннотированных данных, его охвате и пропорциях представительства классов, хорошее разбиение набора на обучающие и тестовые данные, работу с людьми, занимающимися аннотированием, консолидацию решений, контроль качества аннотаторов и аннотирования и многое другое.

1.6. Замечание о терминологии

Термин «признак» используется для обозначения конкретного элемента лингвистических входных данных: слова, суффикса или метки части речи. Например, в частеречном разметчике первого порядка признаками могут быть «текущее слово, предыдущее слово, следующее слово, предыдущая часть речи». Термином «входной вектор» мы обозначаем конкретное значение входных данных. Это не согласуется с основной массой литературы по нейронным сетям, где слово «признак» перегружено, употребляется в обоих случаях, но преимущественно обозначает элемент входного вектора.

1.7. Математическая нотация

Полужирными заглавными буквами обозначаются матрицы ($\boldsymbol{X}$, $\boldsymbol{Y}$, $\boldsymbol{Z}$), полужирными строчными буквами – векторы ($\boldsymbol{b}$). Если имеется последовательность связанных матриц и векторов (например, каждая матрица соответствует одному слою сети), то используются верхние индексы ($\boldsymbol{W}^1$, $\boldsymbol{W}^2$). В тех редких случаях, когда необходимо обозначить степень матрицы или вектора, мы употребляем скобки: $(\boldsymbol{W})^2$; $(\boldsymbol{W}^3)^2$. Квадратные скобки [] используются как оператор взятия индекса для векторов и матриц: $\boldsymbol{b}_{[i]}$ обозначает i-й элемент вектора $\boldsymbol{b}$, а $\boldsymbol{W}_{[i,j]}$ – элемент на пересечении i-го столбца и j-й строки матрицы $\boldsymbol{W}$. Если возникает неоднозначность, то мы иногда прибегаем к более привычной математической нотации и пишем b_i для обозначения i-го элемента вектора $\boldsymbol{b}$ и $w_{i,j}$ для обозначения элементов матрицы $\boldsymbol{W}$. Символом $\cdot$ обозначается оператор скалярного произведения: $\boldsymbol{w} \cdot \boldsymbol{v} = \sum_i w_i v_i = \sum_i \boldsymbol{w}_{[i]} \boldsymbol{v}_{[i]}$. Последовательность векторов $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ обозначается $\boldsymbol{x}_{1:n}$, а последовательность элементов $x_1, \ldots, x_n$ – $x_{1:n}$. Символом $\boldsymbol{x}_{n:1}$ обозначается инвертированная последовательность: $\boldsymbol{x}_{1:n}[i] = \boldsymbol{x}_i, \boldsymbol{x}_{n:1}[i] = \boldsymbol{x}_{n-i+1}$. Конкатенация векторов обозначается $[\boldsymbol{v}_1; \boldsymbol{v}_2]$.

Если явно не оговорено противное, векторы считаются векторами-строками, хотя обычно так не делают. Решение использовать векторы-строки, умножаемые справа на матрицы ($\boldsymbol{xW} + \boldsymbol{b}$), отличается от стандартного – в литературе по нейронным сетям, как правило, используются векторы-столбцы, умножаемые на матрицу слева ($\boldsymbol{Wx} + \boldsymbol{b}$). Мы верим, что читатель сможет приспособиться к нотации векторов-столбцов при чтении литературы[1].

[1] Решение использовать векторы-строки продиктовано следующими соображениями: оно согласуется с тем, как входные векторы и диаграммы сетей обычно изображаются в литературе; иерархическая (слоистая) структура сети становится более наглядной, и вход оказывается самой левой, а не вложенной переменной; размерности полносвязной сети указываются в порядке $d_{in} \times d_{out}$, а не $d_{out} \times d_{in}$; эта нотация лучше согласована с реализацией сетей в коде с использованием библиотек для работы с матрицами, например numpy.

Часть I

КЛАССИФИКАЦИЯ С УЧИТЕЛЕМ И НЕЙРОННЫЕ СЕТИ ПРЯМОГО РАСПРОСТРАНЕНИЯ

Глава 2

Основы обучения и линейные модели

Нейронные сети – тема настоящей книги – это класс алгоритмов машинного обучения с учителем.

В этой главе приводится краткое введение в терминологию и практику машинного обучения с учителем, а также описываются линейные и логлинейные модели бинарной и многоклассовой классификаций.

Здесь же подготавливается почва и определяется нотация, используемая в последующих главах. Читатели, знакомые с линейными моделями, могут сразу перейти к следующим главам, но и им было бы полезно прочитать разделы 2.4 и 2.5.

Теория машинного обучения с учителем и линейные модели – очень обширные темы, так что эту главу никак нельзя считать исчерпывающей. Более полное освещение можно увидеть в книгах Daumé III [2015], Shalev-Shwartz and Ben-David [2014][1] и Mohri et al. [2012].

2.1. Обучение с учителем и параметрические функции

Суть машинного обучения с учителем заключается в создании механизмов, которые могут на основе анализа примеров порождать обобщения. Точнее, вместо того чтобы проектировать алгоритм решения некоторой задачи («отличить спамные сообщения электронной почты от неспамных»), мы проектируем алгоритм, который получает на входе множество помеченных примеров («эта кучка сообщений – спам, а та – не спам»), а на выходе выдает функцию (или программу), которая принимает образец (почтовое сообщение) и возвращает метку (спам или не спам). Ожидается, что результирующая функция будет правильно предсказывать метки даже для образцов, которые не видела в процессе обучения.

Поскольку поиск в множестве всех возможных программ (или всех возможных функций) – невероятно трудная (и к тому же некорректно поставленная) задача, то мы зачастую ограничиваемся поиском по некоторому семейству, например в пространстве всех линейных функций, имеющих d_{in} входов и d_{out} выходов, или в пространстве всех решающих деревьев с d_{in} переменными. Такие семейства

[1] *Шай Шалев-Шварц, Шай Бен-Давид*. Идеи машинного обучения. М.: ДМК Пресс, 2019.

функций называются *классами гипотез*. Ограничившись некоторым классом гипотез, мы вносим в обучаемого *индуктивное смещение* – набор предположений о форме желаемого решения, – но при этом получаем возможность реализовать эффективные процедуры поиска решения. Общий и доступный неподготовленному читателю обзор основных семейств алгоритмов обучения и стоящих за ними предположений *см*. в книге Domingos [2015].

Класс гипотез определяет также, что можно, а что нельзя представить с помощью обучаемого. Один широко распространенный класс гипотез – многомерные линейные функции, т. е. функции вида[1]:

$$f(\boldsymbol{x}) = \boldsymbol{x}\cdot\boldsymbol{W} + \boldsymbol{b}, \qquad \boldsymbol{x} \in \mathbb{R}^{d_{in}}, \quad \boldsymbol{W} \in \mathbb{R}^{d_{in}\times d_{out}}, \quad \boldsymbol{b} \in \mathbb{R}^{d_{out}}. \tag{2.1}$$

Здесь вектор $\boldsymbol{x}$ является *входом* функции, а матрица $\boldsymbol{W}$ и вектор $\boldsymbol{b}$ – *параметрами*. Задача обучаемого – найти такие значения параметров $\boldsymbol{W}$ и $\boldsymbol{b}$, при которых функция ведет себя требуемым образом на наборе входных значений $\boldsymbol{x}_{1:k} = \boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ и соответствующих им выходов $\boldsymbol{y}_{1:k} = \boldsymbol{y}_i, \ldots, \boldsymbol{y}_k$. Таким образом, задача поиска в пространстве функций свелась к задаче поиска в пространстве параметров. Принято обозначать параметры функции буквой Θ. В случае линейной модели $\Theta = \boldsymbol{W}, \boldsymbol{b}$. Иногда нам нужна нотация, в которой параметризация выделена явно, тогда мы включаем параметры в определение функции: $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b}) = \boldsymbol{x}\cdot\boldsymbol{W} + \boldsymbol{b}$.

Как станет ясно из последующих глав, класс гипотез, состоящий из линейных функций, довольно ограниченный, и существует много функций, которые невозможно представить с его помощью (на самом деле он ограничен *линейными* отношениями). С другой стороны, *нейронные сети прямого распространения со скрытыми слоями*, рассматриваемые в главе 4, также являются параметрическими функциями, но образуют очень сильный класс гипотез – они являются *универсальными аппроксиматорами*, т. е. способны представить любую функцию, измеримую по Борелю[2]. Но, несмотря на ограниченность, линейные модели обладают несколькими желательными свойствами: они допускают простое и эффективное обучение, часто приводят к задачам выпуклой оптимизации, обученные модели легко интерпретировать, а в применении они обычно очень эффективны. Линейные и логлинейные модели преобладали в статистической NLP более десяти лет. Ко всему прочему они служат базовыми строительными блоками для более мощных нелинейных сетей прямого распространения, которые мы будем обсуждать в последующих главах.

2.2. Обучающий, тестовый и контрольный наборы

Прежде чем вдаваться в детали линейных моделей, давайте еще раз рассмотрим общую постановку проблемы машинного обучения. У нас имеется набор данных, состоящий из k входных примеров $\boldsymbol{x}_{1:k}$ и соответствующих им золотых меток $\boldsymbol{y}_{1:k}$, а цель заключается в том, чтобы найти функцию $f(\boldsymbol{x})$, которая отображает входы $\boldsymbol{x}$

[1] Как отмечено в разделе 1.7, в этой книге принят неортодоксальный подход, согласно которому векторами считаются *векторы-строки*, а не векторы-столбцы.

[2] *См*. обсуждение в разделе 4.3.

в выходы $\hat{\boldsymbol{y}}$ в соответствии с обучающим набором. Откуда мы знаем, что найденная функция $f()$ действительно хорошая? Можно было бы прогнать обучающие примеры $\boldsymbol{x}_{1:k}$ через $f()$, запомнить ответы $\hat{\boldsymbol{y}}_{1:k}$ и измерить верность. Но этот процесс малоинформативен, ведь наша главная задача – сделать так, чтобы $f()$ хорошо обобщалась на примеры, которые ранее не предъявлялись. Функция $f()$, реализованная в виде таблицы соответствия, т. е. такая, которая ищет вход $\boldsymbol{x}$ в своей памяти и возвращает соответствующее ему значение $\mathbf{y}$ для образцов, которые видела раньше, и случайное значение для всех остальных, идеально пройдет такой тест, но при этом не является хорошей функцией классификации, поскольку вообще не обладает способностью к обобщению. Мы предпочли бы функцию $f()$, которая на некоторых обучающих примерах ошибается, но правильно ведет себя на ранее не предъявлявшихся примерах.

ИСКЛЮЧЕНИЕ ПО ОДНОМУ Мы должны оценить верность обученной функции на образцах, которые она не видела в процессе обучения. Одно из решений – *перекрестная проверка с исключением по одному*: обучить k функций $f_{1:k}$, каждый раз исключая из обучающего набора по одному входному примеру $\boldsymbol{x}_i$, и оценить способность результирующей функции $f_i()$ к предсказанию метки $\boldsymbol{x}_i$. Затем обучить еще одну функцию $f()$ на всем наборе $\boldsymbol{x}_{1:k}$. В предположении, что обучающий набор – это репрезентативная выборка из генеральной совокупности, процентная доля функций $f_i()$, давших правильные предсказания на исключенных примерах, является хорошим приближением к верности $f()$ на новых образцах. Однако эта процедура обходится весьма дорого в плане времени вычислений и используется только в тех случаях, когда количество аннотированных примеров очень мало (порядка сотни). В задачах лингвистической обработки часто встречаются обучающие наборы, содержащие более 10^5 примеров.

ЗАРЕЗЕРВИРОВАННЫЙ НАБОР Более эффективное с точки зрения времени вычислений решение – разбить обучающий набор на два поднабора, скажем, в отношении 80:20, обучить модель на большем поднаборе (*обучающем*), а затем проверить верность на меньшем поднаборе (*зарезервированном*). Это даст разумную оценку верности обученной функции или, по крайней мере, позволит сравнивать качество различных обученных моделей. Но такое решение расточительно в плане использования обучающих примеров. Можно было бы затем переобучить модель на всем наборе. Однако для модели, обученной на заметно большем наборе данных, оценка ошибки, полученная для модели, обученной на меньшем наборе, может оказаться неточной. Но обычно это не проблема, потому что увеличение количества обучающих данных скорее улучшит, а не ухудшит качество предиктора[1].

Разбивать набор следует аккуратно, вообще говоря, лучше предварительно перетасовать примеры в случайном порядке, чтобы гарантировать сбалансированное их распределение между обучающим и зарезервированным набором (так, мы хотим, чтобы доли золотых меток в обоих наборах были примерно одинаковы). Но иногда случайное разбиение не годится: взять, к примеру, ситуацию, когда на вход

[1] Заметим, однако, что некоторые параметры процедуры обучения, в частности скорость обучения и регуляризирующие веса, могут быть чувствительны к размеру обучающего набора, поэтому их настройка на одних данных и последующее переобучение модели с такими настройками на большем наборе данных может привести к неоптимальным результатам.

подаются новостные статьи, собранные за несколько месяцев, и ожидается, что модель будет давать предсказания для новых статей. В данном случае случайное разбиение даст завышенную оценку качества модели: обучающие и зарезервированные примеры будут относиться к одному периоду времени и, значит, к более похожим статьям, что вряд ли будет иметь место на практике. Поэтому нужно включить в обучающий набор старые статьи, а в зарезервированный – статьи поновее, тогда обученная модель будет максимально приближена к реальности.

РАЗБИЕНИЕ НА ТРИ ЧАСТИ Разбиение на обучающий и зарезервированный наборы хорошо работает, если мы обучаем одну модель и хотим оценить ее качество. Но на практике часто обучают несколько моделей, сравнивают их качество и выбирают лучшую. В таком случае разбивать набор на две части недостаточно – выбор лучшей модели, исходя из верности на зарезервированном наборе, дает чрезмерно оптимистичную оценку качества. Мы не знаем, хороши ли выбранные параметры окончательного классификатора в общем случае или только для конкретных примеров из зарезервированного набора. Проблема еще осложняется, если мы производим анализ ошибки, основываясь на зарезервированном наборе, и изменяем признаки или архитектуру модели, ориентируясь на наблюдаемые ошибки. Мы не знаем, будут ли улучшения, основанные на зарезервированных наборах, распространяться на новые образцы. Общепринятая методология – разбивать данные на обучающий, контрольный и тестовый наборы. Таким образом мы получаем два зарезервированных набора: *контрольный* и *тестовый*. Все эксперименты, подстройки, анализ ошибок и выбор модели должны производиться с использованием контрольного набора. После этого единственный прогон окончательной модели на тестовом наборе должен дать хорошую оценку ожидаемого качества на новых образцах. Важно, чтобы тестовый набор оставался нетронутым, на нем должно проводиться как можно меньше экспериментов. Некоторые даже ратуют за то, чтобы разработчик вообще не видел примеров из тестового набора, чтобы они не повлияли на проектирование модели.

2.3. Линейные модели

Договорившись о методологии, мы можем вернуться к описанию линейных моделей для бинарной и многоклассовой классификаций.

2.3.1. Бинарная классификация

В задаче бинарной классификации мы имеем единственный выход, поэтому используем частный случай уравнения (2.1), в котором $d_{out} = 1$ и, стало быть, $\boldsymbol{w}$ – это вектор, а b – скаляр:

$$f(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{w} + b. \tag{2.2}$$

Область значений линейной функции в уравнении (2.2) – интервал $[-\infty, +\infty]$. Чтобы использовать ее для бинарной классификации, обычно передают выход $f(\boldsymbol{x})$ функции sign, которая отображает отрицательные значения в -1 (отрицательный класс), а неотрицательные – в $+1$ (положительный класс).

Рассмотрим задачу предсказания того, в каком из двух районов находится квартира, если известны ее стоимость и площадь. На рис. 2.1 показан график для

нескольких квартир: по оси *x* отложена месячная арендная плата в долларах США, а по оси *y* – площадь в квадратных футах. Синими кружочками обозначен район Дюпон-сёркл в округе Колумбия, а зелеными крестиками – город Фэрфакс в штате Вирджиния. Очевидно, что два района можно разделить прямой линией – квартиры в Дюпон-сёркл дороже квартир той же площади в Фэрфаксе[1]. Этот набор данных *линейно разделим*: два класса можно разделить прямой линией.

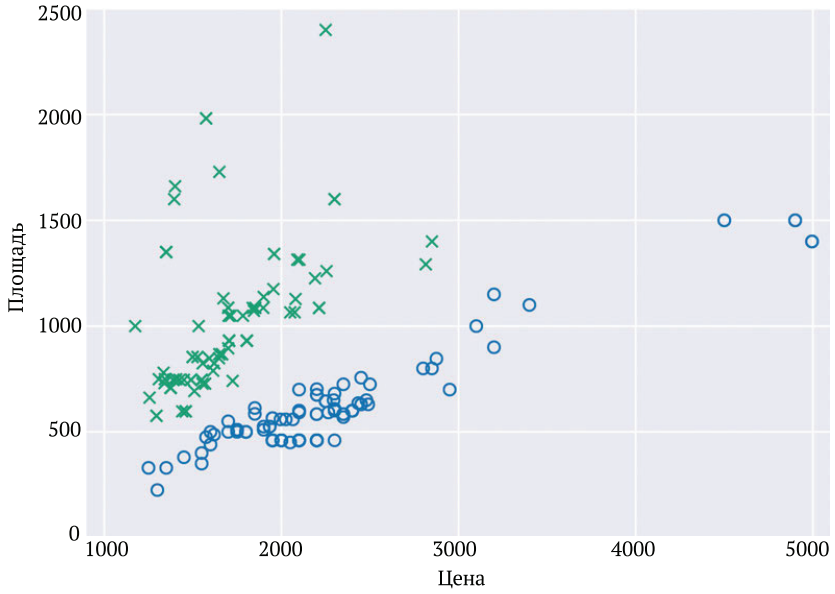


Рис. 2.1 ❖ Данные о квартирах: зависимость арендной платы в долларах США от площади в квадратных футах. *Источник данных*: объявления на сайте Craigslist за период 7–15 июня 2015 года

Каждую точку (квартиру) можно представить двумерным вектором $\boldsymbol{x}$, в котором $\boldsymbol{x}_{[0]}$ – площадь квартиры, а $\boldsymbol{x}_{[1]}$ – ее цена. Тогда получается следующая линейная модель:

$$\begin{aligned}\hat{\boldsymbol{y}} &= \text{sign}(f(\boldsymbol{x})) = \text{sign}(\boldsymbol{x}\cdot\boldsymbol{w} + b)\\ &= \text{sign}(\text{size} \times w_1 + \text{price} \times w_2 + b),\end{aligned}$$

где $\cdot$ обозначает скалярное произведение, b и $\boldsymbol{w} = [w_1; w_2]$ – свободные параметры, и мы предсказываем Фэрфакс, если $\hat{\boldsymbol{y}} \geq 0$, и Дюпон-сёркл в противном случае. Цель обучения – найти такие значения w_1, w_2 и b, чтобы предсказание было правильным для всех наблюдаемых точек[2]. Мы будем обсуждать обучение в разделе 2.7,

[1] Отметим, что анализ только площади или только цены не позволил бы четко провести разделение на две группы.

[2] Геометрически все точки, для которых $\boldsymbol{x}\cdot\boldsymbol{w} + b = 0$ при заданном $\boldsymbol{w}$, образуют *гиперплоскость* (в двумерном пространстве это прямая), разбивающую пространство на две области. Тогда цель обучения состоит в том, чтобы найти такую гиперплоскость, что индуцированная ей классификация правильна.

а пока заметим, что мы ожидаем от процедуры обучения, что она выберет высокое значение w_1 и низкое значение w_2. Обучив модель, мы сможем классифицировать новые точки, подставляя их в это уравнение.

Иногда невозможно разделить точки прямой линией (или, в многомерном пространстве, гиперплоскостью). Такие наборы данных называются *линейно неразделимыми*, они выходят за рамки класса гипотез, состоящего из линейных классификаторов. Для решения проблемы можно либо перейти в пространство более высокой размерности (добавить признаки), либо взять более богатый класс гипотез, либо смириться с неправильной классификацией в некоторых случаях[1].

ПРЕДСТАВЛЕНИЯ ПРИЗНАКОВ В примере выше экспериментальными точками были пары (площадь, цена). Каждое из этих двух свойств считается признаком, по которому классифицируются точки. Это очень удобно, но в большинстве случаев экспериментальные точки задаются в виде реальных объектов, а не напрямую в виде списков признаков. Например, в рассмотренном примере может быть задан список квартир, которые требуется классифицировать. В таком случае мы должны выбрать допускающие измерение свойства квартир, которые, по нашему мнению, могут оказаться полезными признаками для классификации. В нашем примере цена и площадь действительно оказались эффективны. Можно было бы рассмотреть дополнительные свойства, скажем, количество комнат, высоту потолка, покрытие пола, координаты и т. д. Определившись с набором признаков, мы создаем функцию выделения признаков, отображающую реальный объект (т. е. квартиру) в вектор измеримых величин (цена и площадь), который можно подать на вход модели. Выбор признаков критически важен для успеха классификации и определяется информативностью и доступностью признаков (географические координаты были бы гораздо лучшими предикторами местоположения, чем цена и площадь, но, быть может, в нашем распоряжении имеется лишь список состоявшихся сделок, а доступа к географической информации нет). Имея всего два признака, мы легко можем нанести данные на график и посмотреть, нет ли какой-то структуры. Но, как мы увидим в следующем примере, зачастую признаков гораздо больше, поэтому построение графиков и точные рассуждения на практике бесполезны. Главный шаг проектирования линейных моделей, который в этой книге мы по большей части игнорируем, – это построение функции выделения признаков (конструирование признаков). Одно из обещаний глубокого обучения заключается в том, что оно существенно упрощает процесс конструирования признаков: проектировщик модели задает лишь небольшой набор базовых, или «естественных», признаков, обучаемая архитектура нейронной сети сама комбинирует их, строя более значимые высокоуровневые признаки, или представления. Но все равно нужно выбрать подходящий набор базовых признаков и включить их в подходящую архитектуру. Обсуждение типичных признаков для текстовых данных мы отложим до глав 6 и 7.

[1] Неправильная классификация некоторых примеров иногда оказывается хорошей идеей. Например, если у нас есть основания полагать, что некоторые экспериментальные точки являются *выбросами* – примерами, которые принадлежат одному классу, но по ошибке помечены другим классом.

Обычно признаков гораздо больше, чем два. Рассмотрим задачу о том, как различить документы, написанные на английском и на немецком языках. Оказывается, что для ее решения вполне подойдут частоты букв в качестве предикторов (признаков). А еще более информативными являются счетчик *биграмм*, т. е. пар соседних букв[1]. В предположении, что в алфавите 28 букв (a–z, пробел и специальный символ для всех остальных букв, включая цифры, знаки препинания и т. д.), документ можно представить в виде вектора в пространстве размерности 28×28, $\boldsymbol{x} \in \mathrm{R}^{784}$, где элемент $\boldsymbol{x}_{[i]}$ равен количеству данной комбинации букв в документе, нормированному на длину документа. Например, если обозначить x_{ab} элемент $\boldsymbol{x}$, соответствующий биграмме ab, то:

$$x_{ab} = \frac{\#_{ab}}{|D|}, \tag{2.3}$$

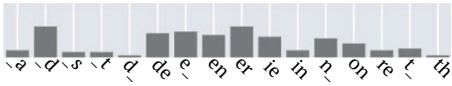
где $\#_{ab}$ – сколько раз биграмма ab встречается в документе, а $|D|$ – общее число биграмм в документе (длина документа).

Рис. 2.2 ❖ Буквенные биграммы для документов на английском (слева) и немецком (справа) языках. Знаками подчеркивания обозначены пробелы

На рис. 2.2 показаны гистограммы распределения биграмм в нескольких текстах на английском и немецком языках. Для удобства приведены только самые часто встречающиеся символьные биграммы, а не векторы признаков целиком. Слева

[1] Может показаться, что хорошими предикторами были бы также *слова*, но буквы и буквенные биграммы гораздо надежнее: мы вполне можем встретить новый документ, в котором нет ни одного слова, вошедшего в обучающий набор, тогда как документы, в которых нет ни одной различающей буквенной биграммы, встречаются гораздо реже.

мы видим биграммы для английских текстов, а справа – для немецких. В данных имеются очевидные паттерны, поэтому, получив новый текст, например:



мы, вероятно, сможем сказать, что он больше похож на немецкий, чем на английский. Заметим, однако, что не существует какого-то одного твердого правила, скажем, «если встречается th, то это английский текст» или «если встречается ie, то это немецкий текст»: хотя в немецких текстах комбинация букв th встречается значительно реже, чем в английских, она все же возможна, как и комбинация ie в английских текстах. Для принятия решения необходимо взвешивать различные факторы. Давайте формализуем постановку задачи в машинном обучении.

Мы снова можем воспользоваться линейной моделью:

$$\begin{aligned}\hat{\boldsymbol{y}} = \text{sign}(f(\boldsymbol{x})) &= \text{sign}(\boldsymbol{x}\cdot\boldsymbol{w} + b) \\ &= \text{sign}(x_{aa} \times w_{aa} + x_{ab} \times w_{ab} + x_{ac} \times w_{ac} \ldots + b).\end{aligned} \tag{2.4}$$

Документ будем считать английским, если $f(\boldsymbol{x}) \geq 0$, и немецким – в противном случае. Интуитивно понятно, что обучение должно сопоставить большие положительные значения элементам $\boldsymbol{w}$, ассоциированным с парами букв, которые гораздо чаще встречаются в английских текстах, чем в немецких (например, th), большие отрицательные значения – парам букв, которые гораздо чаще встречаются в немецких текстах, чем в английских (ie, en), и значения, близкие к нулю, – парам букв, которые одинаково часто или редко встречаются в обоих языках.

Заметим, что, в отличие от двумерного случая с домами (зависимость цены от площади), здесь не получится наглядно представить точки и решающую границу, поэтому геометрическая интуиция не дает очевидного ответа. Вообще, большинству людей трудно рассуждать о геометрии пространств, размерность которых больше трех, так что рекомендуется интерпретировать линейные модели в терминах назначения весов признакам – это проще представить и обсуждать.

2.3.2. Логлинейная бинарная классификация

Значение функции $f(\boldsymbol{x})$ принадлежит диапазону $[-\infty, +\infty]$, а мы хотим отобразить его в один из двух классов $\{-1, +1\}$, пользуясь функцией *sign*. Эта функция годится, если нас интересует только назначенный класс. Но иногда интерес представляет также степень уверенности в решении или вероятность того, что классификатор отнесет образец к определенному классу. Чтобы добиться этого, мы можем отображать значение функции в диапазон [0, 1], воспользовавшись сплющивающей функцией типа сигмоиды $\sigma(x) = 1/(1 + e^{-x})$:

$$\hat{y} = \sigma(f(\boldsymbol{x})) = \frac{1}{1+e^{-(\boldsymbol{x}\cdot\boldsymbol{w}+b)}}. \tag{2.5}$$

На рис. 2.3 показан график сигмоиды. Она монотонно возрастает и отображает значения в диапазон [0, 1][1], причем 0 отображается в ½. При использовании совместно с подходящей *функцией потерь* (*см.* раздел 2.7.1) бинарные предсказания, сделанные с помощью логлинейной модели, можно интерпретировать как оценки вероятности принадлежности к классам: $\sigma(f(\boldsymbol{x})) = P(\hat{\boldsymbol{y}} = 1 \mid \boldsymbol{x})$ – вероятность того, что $\boldsymbol{x}$ принадлежит положительному классу. Аналогично $P(\hat{\boldsymbol{y}} = 0 \mid \boldsymbol{x}) = 1 - P(\hat{\boldsymbol{y}} = 1 \mid \boldsymbol{x}) = 1 - \sigma(f(\boldsymbol{x}))$. Чем значение ближе к 0 или к 1, тем больше уверенность модели в предсказании, а значение 0.5 означает, что модель ни в чем не уверена.

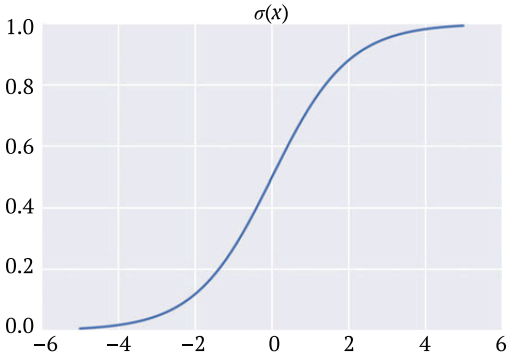


Рис. 2.3 ❖ Сигмоидная функция $\sigma(x)$

2.3.3. Многоклассовая классификация

Выше мы видели примеры *бинарной классификации*, когда возможных классов всего два. У бинарной классификации есть место в жизни, но большинство задач классификации *многоклассовые*, т. е. требуется отнести образец к одному из k различных классов. Например, нам могут предъявить документ и попросить отнести его к одному из шести языков: *английскому*, *французскому*, *немецкому*, *итальянскому*, *испанскому*, *прочему*. Одно из возможных решений – рассмотреть шесть векторов весов $\boldsymbol{w}^{\text{En}}, \boldsymbol{w}^{\text{Fr}}, \ldots$ и шесть смещений – по одному для каждого языка – и в качестве предсказания вернуть результат с наибольшей оценкой[2]:

$$\hat{y} = f(\boldsymbol{x}) = \underset{L \in \{\text{En,Fr,Gr,It,Sp,O}\}}{\operatorname{argmax}} \boldsymbol{x} \cdot \boldsymbol{w}^L + b^L. \tag{2.6}$$

Шесть наборов параметров $\boldsymbol{w}^L \in \mathbb{R}^{784}$, b^L можно организовать в виде матрицы $\boldsymbol{W} \in \mathbb{R}^{784\times 6}$ и вектора $\boldsymbol{b} \in \mathbb{R}^6$, а приведенное выше равенство переписать в виде:

$$\begin{aligned} \hat{\boldsymbol{y}} &= f(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{W} + \boldsymbol{b}; \\ \text{prediction} &= \hat{y} = \underset{i}{\operatorname{argmax}}\, \hat{\boldsymbol{y}}_{[i]}. \end{aligned} \tag{2.7}$$

---

[1] В математической литературе принято обозначать открытые интервалы круглыми скобками, а замкнутые – квадратными, но мы сохраняем нотацию автора. – *Прим. перев.*

[2] Существует много способов смоделировать многоклассовую классификацию, в том числе сведение к бинарной. Их рассмотрение выходит за рамки этой книги, но хороший обзор можно найти в работе Allwein et al. [2000].

Здесь $\hat{\boldsymbol{y}} \in \mathrm{R}^6$ – вектор оценок, назначенных моделью каждому языку, а предсказанный язык мы получаем путем применения функции argmax к элементам $\hat{\boldsymbol{y}}$.

2.4. Представления

Вектор $\hat{\boldsymbol{y}}$, получающийся в результате применения уравнения 2.7 обученной модели к документу, можно рассматривать как *представление* документа, улавливающее важные для нас свойства, а именно оценки различных языков. Представление $\hat{\boldsymbol{y}}$ содержит больше информации, чем само предсказание $\hat{\boldsymbol{y}} = \mathrm{argmax}_i \hat{\boldsymbol{y}}_{[i]}$: например, $\hat{\boldsymbol{y}}$ можно использовать, чтобы выделить документы, написанные в основном на немецком, но содержащие довольно много французских слов. Путем кластеризации документов на основе их векторных представлений мы, возможно, могли бы найти документы, написанные на региональных диалектах или на нескольких языках.

Векторы $\boldsymbol{x}$, содержащие нормированные счетчики буквенных биграмм в документах, также являются представлениями документов, содержащими примерно такую же информацию, как векторы $\hat{\boldsymbol{y}}$. Однако представления в виде $\hat{\boldsymbol{y}}$ более компактны (6 элементов вместо 784) и специализированы для задачи предсказания языка (кластеризация по векторам $\boldsymbol{x}$, вероятно, выявила бы сходство документов, связанное не с конкретной смесью языков, а с темой документа или стилем написания).

Можно считать, что и обученная матрица $\boldsymbol{W} \in \mathbb{R}^{784\times 6}$ содержит обученные представления. Как показано на рис. 2.4, мы можем рассматривать матрицу $\boldsymbol{W}$ двумя способами: по строкам и по столбцам. Каждый из шести столбцов $\boldsymbol{W}$ соответствует одному языку и может считаться 784-мерным векторным представлением этого языка в терминах характерных для него буквенных биграмм. Шесть векторов языка мы можем кластеризовать по сходству. С другой стороны, каждая из 784 строк $\boldsymbol{W}$ соответствует одной буквенной биграмме и дает шестимерное векторное представление этой биграммы в терминах возможных языков.

Представления играют главную роль в глубоком обучении. Можно даже утверждать, что вся сила глубокого обучения заключается в способности находить хорошие представления. В линейном случае представления допускают интерпретацию в том смысле, что можно дать наглядную интерпретацию каждому измерению представляющего вектора (например, измерение соответствует языку или буквенной биграмме). В общем случае это уже не так – модели глубокого обучения части обучают каскад представлений входных данных, в котором каждое следующее строится на основе предыдущего. Цель при этом – наилучшим образом смоделировать поставленную задачу, но сами представления невозможно интерпретировать, т. е. мы не знаем, какие свойства входных данных они улавливают. Тем не менее эти представления очень полезны для предсказания. Более того, на границах модели, т. е. на входе и на выходе, мы получаем представления, соответствующие определенным аспектам входных данных (т. е. векторное представление каждой буквенной биграммы) или выходных данных (т. е. векторное представление каждого выходного класса). Мы вернемся к этому вопросу в разделе 8.3, после того как обсудим нейронные сети и кодирование категориальных признаков в виде плотных векторов. Рекомендуется вернуться к этому месту после прочтения раздела 8.3.

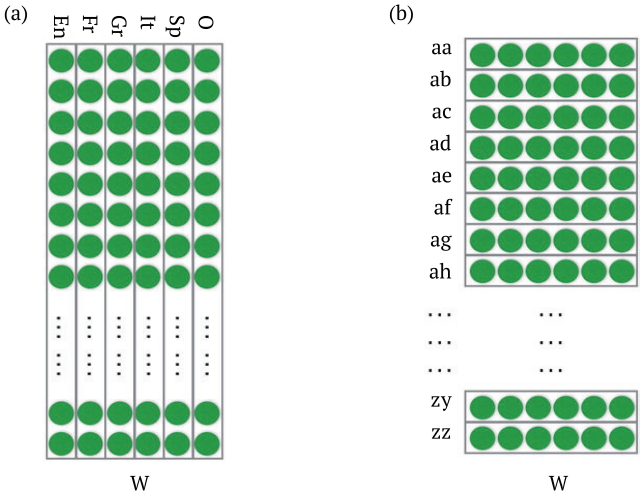


Рис. 2.4 ❖ Два взгляда на матрицу $\boldsymbol{W}$:
(a) каждый столбец соответствует языку;
(b) каждая строка соответствует буквенной биграмме

2.5. Представления в виде унитарного и плотного векторов

Входной вектор $\boldsymbol{x}$ в примере о классификации по языку содержит нормированные счетчики биграмм в документе D. Этот вектор можно представить в виде среднего значения $|D|$ векторов, каждый из которых соответствует документу в позиции i:

$$\boldsymbol{x} = \frac{1}{|D|}\sum_{i=1}^{|D|} \boldsymbol{x}^{D_{[i]}}, \tag{2.8}$$

где $D_{[i]}$ – биграмма в i-м документе, а каждый вектор $\boldsymbol{x}^{D_{[i]}} \in \mathbb{R}^{784}$ *унитарный*, т. е. все его элементы, кроме одного, равны нулю, а элемент, соответствующий буквенной биграмме $D_{[i]}$, равен 1.

Получающийся таким образом вектор $\boldsymbol{x}$ обычно называют *усредненным мешком биграмм* (в более общем случае *усредненным мешком слов*, или просто *мешком слов*). Представление в виде мешка слов (bag-of-words – BOW) содержит сведения об идентификаторах всех «слов» (в данном случае биграмм), встречающихся в документе, без учета их порядка. Унитарное представление можно рассматривать как мешок с одним словом.

Взгляд на строки матрицы $\boldsymbol{W}$ как на представления буквенных биграмм наводит на мысль об альтернативном способе вычисления вектора представления документа $\hat{\boldsymbol{y}}$ в уравнении (2.7). Обозначив $\boldsymbol{W}^{D_{[i]}}$ строку $\boldsymbol{W}$, соответствующую биграмме $D_{[i]}$, мы сможем взять в качестве представления $\boldsymbol{y}$ документа D усреднение представлений буквенных биграмм, встречающихся в документе:

$$\hat{\boldsymbol{y}} = \frac{1}{|D|}\sum_{i=1}^{|D|} \boldsymbol{W}^{D_{[i]}}. \tag{2.9}$$

Это представление часто называют *непрерывным мешком слов* (continuous bag of words – CBOW), поскольку оно состоит из суммы представлений слов, где представление каждого «слова» – непрерывный вектор малой размерности.

Заметим, что уравнение (2.9) и член $\boldsymbol{x} \cdot \boldsymbol{W}$ в уравнении (2.7) эквивалентны. Убедиться в этом просто:

$$\begin{aligned} \boldsymbol{y} &= \boldsymbol{x} \cdot \boldsymbol{W} \\ &= \left(\frac{1}{|D|} \sum_{i=1}^{|D|} \boldsymbol{x}^{D_{[i]}} \right) \cdot \boldsymbol{W} \\ &= \frac{1}{|D|} \sum_{i=1}^{|D|} (\boldsymbol{x}^{D_{[i]}} \cdot \boldsymbol{W}) \\ &= \frac{1}{|D|} \sum_{i=1}^{|D|} \boldsymbol{W}^{D_{[i]}}. \end{aligned} \tag{2.10}$$

Иными словами, представление в виде непрерывного мешка слов (CBOW) можно получить либо суммированием векторов представлений слов, либо умножением вектора мешка слов на матрицу, строки которой соответствуют плотному представлению слов (такие матрицы называют также *матрицами погружения*). Мы вернемся к этой теме в главе 8 (конкретно в разделе 8.3), когда будем обсуждать представления признаков в моделях глубокого обучения для текста.

2.6. Логлинейная многоклассовая классификация

В бинарном случае мы преобразовывали линейное предсказание в оценку вероятности, передавая его сигмоидной функции, так получилась логлинейная модель. В многоклассовом случае аналогом является передача вектора оценок функции *softmax*:

$$\text{softmax}(x)_{[i]} = \frac{e^{\boldsymbol{x}_{[i]}}}{\sum_j e^{\boldsymbol{x}_{[j]}}}, \tag{2.11}$$

что дает

$$\begin{aligned} \hat{\boldsymbol{y}} &= \text{softmax}(\boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}) \\ \hat{\boldsymbol{y}}_{[i]} &= \frac{e^{(\boldsymbol{x}\boldsymbol{W}+\boldsymbol{b})_{[i]}}}{\sum_j e^{(\boldsymbol{x}\boldsymbol{W}+\boldsymbol{b})_{[j]}}}. \end{aligned} \tag{2.12}$$

Функция *softmax* преобразует $\boldsymbol{x}$ в вектор $\hat{\boldsymbol{y}}$, все элементы которого положительны и в сумме равны 1, так что их можно интерпретировать как распределение вероятностей.

2.7. Обучение как оптимизация

Напомним, что на вход алгоритма обучения с учителем подается *обучающий набор* из n примеров $x_{1:n} = x_1, x_2, \ldots, x_n$ вместе с соответствующими метками $y_{1:n} = y_1, y_2,$

..., y_n. Без ограничения общности можно предположить, что желаемыми входами и выходами являются векторы $\boldsymbol{x}_{1:n}$, $\boldsymbol{y}_{1:n}$[1].

Цель алгоритма – вернуть функцию $f()$, которая верно отображает входные примеры на их метки, т. е. такую функцию $f()$, что предсказания $\hat{\boldsymbol{y}} = f(x)$ на обучающем наборе верны. Чтобы уточнить постановку задачи, введем понятие *функции потерь*, которая количественно выражает потерю, вызванную предсказанием $\hat{\boldsymbol{y}}$ вместо истинных меток $\boldsymbol{y}$. Формально функция потерь $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ сопоставляет числовую оценку (скаляр) предсказанному выходу $\hat{\boldsymbol{y}}$, если известен истинный выход $\boldsymbol{y}$. Функция потерь должна быть ограничена снизу, причем минимум должен достигаться только в случае, когда предсказание правильно.

Параметры обученной функции (матрица $\boldsymbol{W}$ и вектор смещений $\boldsymbol{b}$) устанавливаются так, чтобы минимизировать потерю L на обучающих примерах (обычно минимизируется сумма потерь по всем примерам).

Конкретно, пусть даны помеченный обучающий набор ($\boldsymbol{x}_{1:n}$, $\boldsymbol{y}_{1:n}$), значение функции L для каждого примера и параметрическая функция $f(\boldsymbol{x}; \Theta)$. Тогда потеря на всем корпусе относительно параметров Θ определяется как средняя потеря по всем обучающим примерам:

$$\mathcal{L}(\Theta) = \frac{1}{n}\sum_{i=1}^{n} L(f(\boldsymbol{x}_i; \Theta), \boldsymbol{y}_i). \tag{2.13}$$

При таком подходе обучающие примеры фиксированы, а потеря определяется значениями параметров. Цель алгоритма обучения – найти значения параметров, доставляющие минимум функции $\mathcal{L}$:

$$\hat{\Theta} = \underset{\Theta}{\operatorname{argmin}} \mathcal{L}(\Theta) = \underset{\Theta}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n} L(f(\boldsymbol{x}_i; \Theta), \boldsymbol{y}_i). \tag{2.14}$$

Формула (2.14) – это попытка минимизировать потерю любой ценой, что может привести к *переобучению* на обучающих данных. Чтобы воспрепятствовать этому, часто устанавливаются мягкие ограничения на форму решения. Для этого используется функция $R(\Theta)$, которая принимает параметры и возвращает скаляр, отражающий их «сложность», которую мы стремимся уменьшить. Прибавляя R к целевой функции, мы получаем задачу оптимизации баланса между низкой потерей и низкой сложностью:

$$\hat{\Theta} = \underset{\Theta}{\operatorname{argmin}} \left(\overbrace{\frac{1}{n}\sum_{i=1}^{n} L(f(\boldsymbol{x}_i; \Theta), \boldsymbol{y}_i)}^{\text{потери}} + \overbrace{\lambda R(\Theta)}^{\text{регуляризация}} \right). \tag{2.15}$$

Функция R называется *регуляризирующим членом*. Различные комбинации функций потерь и регуляризирующих членов дают различные алгоритмы обучения с разными индуктивными смещениями.

---

[1] Во многих случаях ожидаемый выход естественно считать скаляром (назначенным классом), а не вектором. Тогда $\boldsymbol{y}$ – это просто соответствующий унитарный вектор, а $\operatorname{argmax}_i \boldsymbol{y}_{[i]}$ – соответствующий ему класс.

Теперь обратимся к обсуждению наиболее распространенных функций потерь (раздел 2.7.1), а также регуляризации и регуляризаторов (раздел 2.7.2). Затем в разделе 2.8 мы представим алгоритм для решения задачи минимизации (уравнение (2.15)).

2.7.1. Функции потерь

В качестве функции потерь можно взять произвольную функцию, отображающую два вектора в скаляр. На практике для целей оптимизации мы ограничиваемся функциями, для которых легко вычислить градиенты (или субградиенты)[1]. В большинстве случаев достаточно и рекомендуется выбирать хорошо известную функцию потерь, а не изобретать свою собственную. Детальное обсуждение функций потерь для бинарной классификации и их теоретическое обоснование см. в работе Zhang [2004]. Ниже мы рассмотрим некоторые функции потерь, которые часто используются в сочетании с линейными моделями и нейронными сетями в NLP.

КУСОЧНО-ЛИНЕЙНАЯ (БИНАРНАЯ) В проблемах бинарной классификации выходом классификатора является один скаляр $\tilde{y}$, а желаемый выход – число y, принадлежащее множеству $\{+1,-1\}$. Правило классификации имеет вид $\hat{\mathbf{y}} = \text{sign}(\tilde{y})$, и классификация считается правильной, если $y \cdot \tilde{y} > 0$, т. е. знаки y и $\tilde{y}$ совпадают. Кусочно-линейная функция потерь, которую также называют потерей зазора или SVM-потерей, определяется следующим образом:

$$L_{\text{hinge(binary)}}(\tilde{y}, y) = \max(0, 1 - y \cdot \tilde{y}). \tag{2.16}$$

Эта потеря равна 0, если знаки y и $\tilde{y}$ совпадают и $|\tilde{y}| \geq 1$. В остальных случаях функция потерь линейна. Иными словами, бинарная кусочно-линейная потеря – это попытка добиться правильной классификации с *зазором*, не меньшим 1.

КУСОЧНО-ЛИНЕЙНАЯ (МНОГОКЛАССОВАЯ) Кусочно-линейная функция потерь была обобщена на несколько классов в работе Crammer and Singer [2002]. Пусть $\hat{\mathbf{y}} = \hat{\mathbf{y}}_{[1]}, \ldots, \hat{\mathbf{y}}_{[n]}$ – выходной вектор классификатора, а $\mathbf{y}$ – унитарный вектор, соответствующий правильному выходному классу. Правило классификации определяется как выбор класса с наибольшей оценкой:

$$\text{prediction} = \underset{i}{\text{argmin}}\, \hat{\mathbf{y}}_{[i]}. \tag{2.17}$$

Обозначим $t = \text{argmax}_i \mathbf{y}_{[i]}$ правильный класс, а $k = \text{argmax}_{i \neq t} \hat{\mathbf{y}}_{[i]}$ – класс с наибольшей оценкой, такой что $k \neq t$. Тогда многоклассовая кусочно-линейная функция потерь определяется следующим образом:

$$L_{\text{hinge(multi-class)}}(\hat{\mathbf{y}}, \mathbf{y}) = \max(0, 1 - (\hat{\mathbf{y}}_{[t]} - \hat{\mathbf{y}}_{[k]})). \tag{2.18}$$

Многоклассовая кусочно-линейная потеря – это попытка назначить правильному классу оценку, большую оценок всех остальных классов на зазор, равный, как минимум, 1.

---

[1] Градиент функции k переменных – это набор k частных производных по каждой переменной. Мы будем подробнее обсуждать градиенты в разделе 2.8.

Как бинарная, так и многоклассовая кусочно-линейная потеря предназначена для использования в сочетании с линейными выходами. Кусочно-линейная потеря полезна, когда требуется однозначное решающее правило, а не вероятности принадлежности к классам.

ЛОГАРИФМИЧЕСКАЯ ПОТЕРЯ – это широко распространенная вариация на тему кусочно-линейной потери, которую можно рассматривать как «мягкую» версию последней с бесконечным зазором [LeCun et al., 2006]:

$$L_{\log}(\hat{\mathbf{y}}, \mathbf{y}) = \log(1 + \exp(-(\hat{\mathbf{y}}_{[t]} - \hat{\mathbf{y}}_{[k]})). \tag{2.19}$$

БИНАРНАЯ ПЕРЕКРЕСТНАЯ ЭНТРОПИЯ Функция потерь, называемая бинарной перекрестной энтропией, или *логистической потерей*, используется при бинарной классификации с условной вероятностью на выходе. Предположим, что имеется два целевых класса, обозначаемых 0 и 1, т. е. правильная метка $y \in \{0, 1\}$. Выход классификатора $\tilde{y}$ передается сигмоидной функции (она называется также логистической) $\sigma(x) = 1/(1 + e^{-x})$, которая отображает его в диапазон [0, 1], и результат интерпретируется как условная вероятность $\hat{\mathbf{y}} = \sigma(\tilde{y}) = P(y = 1|x)$. Правило предсказания имеет вид:

$$\text{prediction} = \begin{cases} 0 & \hat{y} < 0.5 \\ 1 & \hat{y} \geq 0.5 \end{cases}.$$

Сеть обучается максимизировать логарифмическую условную вероятность $\log P(y = 1|x)$ для каждого обучающего примера (x, y). Логистическая потеря определяется следующим образом:

$$L_{\text{logistic}}(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}). \tag{2.20}$$

Логистическая потеря полезна, когда мы хотим, чтобы сеть порождала условную вероятность класса для проблемы бинарной классификации. При этом предполагается, что выходной слой преобразуется сигмоидной функцией.

КАТЕГОРИАЛЬНАЯ ПЕРЕКРЕСТНАЯ ЭНТРОПИЯ Функция потерь, называемая категориальной перекрестной энтропией (или *отрицательным логарифмическим правдоподобием*), используется, когда требуется интерпретировать оценки как вероятности.

Пусть $\mathbf{y} = \mathbf{y}_{[1]}, \ldots, \mathbf{y}_{[n]}$ – вектор, представляющий истинное мультиномиальное распределение меток $1, \ldots, n$[1], и пусть $\hat{\mathbf{y}} = \hat{\mathbf{y}}_{[1]}, \ldots, \hat{\mathbf{y}}_{[n]}$ – выход линейного классификатора, преобразованный функцией softmax (*см.* раздел 2.6), который представляет условное распределение вероятностей принадлежности к классам $\hat{\mathbf{y}}_{[i]} = P(y = i|x)$. Категориальная перекрестная энтропия измеряет степень непохожести истинного распределения меток $\mathbf{y}$ и предсказанного распределения $\hat{\mathbf{y}}$, она определяется следующим образом:

$$L_{\text{cross-entropy}}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_i \mathbf{y}_{[i]} \log(\hat{\mathbf{y}}_{[i]}). \tag{2.21}$$

[1] В этой формулировке предполагается, что образец может с некоторой вероятностью принадлежать сразу нескольким классам.

Для задач однозначной классификации, когда каждому обучающему примеру сопоставляется единственный класс, $\boldsymbol{y}$ – унитарный вектор, соответствующий истинному классу. В таких случаях определение перекрестной энтропии можно упростить:

$$L_{\text{cross-entropy(hard classification)}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\log(\hat{\boldsymbol{y}}_{[t]}), \tag{2.22}$$

где t – правильный класс. Это попытка сделать массу вероятности, сопоставленную правильному классу t, равной 1. Поскольку softmax преобразует оценки $\hat{\boldsymbol{y}}$ в неотрицательные числа, в сумме составляющие 1, увеличение массы, сопоставленной правильному классу, означает уменьшение массы, сопоставленной всем остальным классам.

Перекрестная энтропия в качестве потери очень часто встречается в логлинейных моделях и в литературе по нейронным сетям, поскольку порождает многоклассовый классификатор, который предсказывает не только метку наилучшего класса, но и распределение всех остальных меток. При этом предполагается, что выход классификатора преобразуется функцией softmax.

ПОТЕРИ РАНЖИРОВАНИЯ В некоторых случаях учитель получает не метки, а пары, состоящие из правильного объекта $\boldsymbol{x}$ и неправильного $\boldsymbol{x}'$, а его цель – назначить правильным объектам более высокую оценку, чем неправильным. Такие ситуации возникают, когда мы располагаем только положительными примерами и генерируем отрицательные путем искажения положительных. Тогда полезна потеря, ранжирующая на основе зазора, которая определяется для пары примеров следующим образом:

$$L_{\text{ranking(margin)}}(\boldsymbol{x}, \boldsymbol{x}') = \max(0, 1 - (f(\boldsymbol{x}) - (f(\boldsymbol{x}')))), \tag{2.23}$$

где $f(\boldsymbol{x})$ – оценка, которую классификатор назначает входному вектору $\boldsymbol{x}$. Цель заключается в том, чтобы зазор между оценками (рангами) правильных и неправильных входных примеров составлял не меньше 1.

Часто используется также логарифмический вариант потери ранжирования:

$$L_{\text{ranking(log)}}(\boldsymbol{x}, \boldsymbol{x}') = \log(1 + \exp(-(f(\boldsymbol{x}) - (f(\boldsymbol{x}'))))). \tag{2.24}$$

В качестве примера использования потери ранжирования в задачах лингвистической обработки приведем обучение для вывода предобученного погружения слов (*см.* раздел 10.4.2), когда дана последовательность правильных слов и последовательность искаженных слов, а цель состоит в том, чтобы ранжировать правильную последовательность выше, чем неправильную [Collobert and Weston, 2008]. Аналогично в работе Van de Cruys [2014] потеря ранжирования используется в задаче о сочетаемостном предпочтении термов, когда сеть обучается ранжировать правильные пары глагол–дополнение выше, чем автоматически порожденные неправильные, а в работе Weston et al. [2013] была обучена модель, которая ранжирует правильные тройки (голова, отношение, хвост) выше неправильных в задаче об извлечении информации. Пример использования логарифмической потери ранжирования можно найти в работе Gao et al. [2014]. Вариации логарифмической потери ранжирования с различными зазорами между положительным и отрицательным классами описаны в работе dos Santos et al. [2015].

2.7.2. Регуляризация

Рассмотрим задачу оптимизации в уравнении (2.14). У нее может быть много решений и возможно переобучение, особенно если размерность высока. Возьмем в качестве примера задачу об определении языка и предположим, что один из документов в обучающем наборе (назовем его $\boldsymbol{x}_o$) является выбросом: он написан на немецком языке, но ошибочно помечен как французский. Чтобы уменьшить потерю, обучаемый может выявить в $\boldsymbol{x}_o$ признаки (буквенные биграммы), которые встречаются только в небольшом количестве других документов, и назначить им очень сильные веса для отнесения (неправильного) к классу «французский». Тогда в других немецких документах, в которых встречаются эти признаки и которые теперь ошибочно классифицируются как французские, обучаемый увидит другие немецкие биграммы и поднимет их веса, чтобы эти документы снова были классифицированы как немецкие. Это решение проблемы обучения плохое, поскольку оно обучается поступать неправильно, и может получиться так, что тестовые немецкие документы, имеющие много общих слов с $\boldsymbol{x}_o$, будут ошибочно классифицированы как французские. Интуитивно понятно, что в таких случаях мы хотели бы увести обучаемого от подобных неправильных решений в сторону более естественных, когда считается приемлемой неправильная классификация небольшого количества примеров, если они плохо согласуются с остальными.

Достигается это путем добавления *регуляризирующего члена* R в целевую функцию задачи оптимизации. Этот член управляет сложностью параметров и позволяет избежать переобучения:

$$\begin{aligned}\hat{\Theta} &= \underset{\Theta}{\operatorname{argmin}} \mathcal{L}(\Theta) + \lambda R(\Theta) \\ &= \underset{\Theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} L(f(\boldsymbol{x}_i; \Theta), \boldsymbol{y}_i) + \lambda R(\Theta).\end{aligned} \tag{2.25}$$

Регуляризирующий член учитывает значения параметров и оценивает их сложность. Затем мы ищем такие значения параметров, для которых одновременно достигается низкая потеря и низкая сложность. Гиперпараметр[1] λ служит для управления степенью регуляризации: предпочитаем ли мы простую модель модели с низкой потерей или наоборот. Значение λ задается вручную, исходя из качества классификации на контрольном наборе. Хотя в уравнении (2.25) одна регуляризирующая функция и значение λ указываются сразу для всех параметров, можно, конечно, иметь разные регуляризаторы для каждого элемента Θ.

На практике регуляризаторы R ставят знак равенства между сложностью и большими весами и стремятся к тому, чтобы значения параметров были как можно меньше. В частности, регуляризаторы R вычисляют нормы матриц параметров и подталкивают обучаемого в сторону решений с низкой нормой. Часто в качестве R выбирают норму L_2, норму L_1 или эластичную сеть.

L_2-РЕГУЛЯРИЗАЦИЯ В случае L_2-регуляризации R – это квадрат нормы L_2 матрицы параметров, т. е. производится попытка уменьшить сумму квадратов значений параметров:

[1] *Гиперпараметром* называется параметр модели, который не обучается в процессе оптимизации, а должен быть задан вручную.

$$R_{L_2}(\boldsymbol{W}) = \|\boldsymbol{W}\|_2^2 = \sum_{i,j}(\boldsymbol{W}_{[i,j]})^2. \tag{2.26}$$

L_2-регуляризатор также называют *априорным гауссовым распределением,* или *снижением весов*. Заметим, что L_2-регуляризированные модели сильно штрафуются за высокие значения параметров, но если значение достаточно близко к нулю, то эффектом регуляризации можно пренебречь. Модель предпочтет уменьшить значение одного параметра с высоким весом на 1, чем значения каждого из десяти параметров, уже имеющих низкие веса, на 0,1.

L_1-РЕГУЛЯРИЗАЦИЯ В этом случае в качестве R берется норма L_1 матрицы параметров, т. е. производится попытка уменьшить сумму абсолютных величин параметров:

$$R_{L_1}(\boldsymbol{W}) = \|\boldsymbol{W}\|_1 = \sum_{i,j}|\boldsymbol{W}_{[i,j]}|. \tag{2.27}$$

В противоположность L_2 L_1-регуляризатор одинаково штрафует за высокие и низкие значения, стремясь сделать все ненулевые параметры ближе нулю. Поэтому он поощряет разреженные решения – модели, в которых многие параметры равны нулю. L_1-регуляризатор называют также *разреженным априорным распределением*, или *lasso*[1] [Tibshirani, 1994].

ЭЛАСТИЧНАЯ СЕТЬ В регуляризации методом эластичной сети [Zou and Hastie, 2005] сочетаются регуляризация по норме L_1 и по норме L_2:

$$R_{\text{elastic-net}}(\boldsymbol{W}) = \lambda_1 R_{L_1}(\boldsymbol{W}) + \lambda_2 R_{L_2}(\boldsymbol{W}). \tag{2.28}$$

Прореживание. Еще одна форма регуляризации, очень эффективная в нейронных сетях, называется *прореживанием* (dropout). Мы обсудим ее в разделе 4.6.

2.8. Градиентная оптимизация

Для обучения модели мы должны решить задачу оптимизации (2.25). Обычно для этого используется метод на основе градиента. Грубо говоря, градиентные методы состоят в повторяющемся вычислении оценки потери $\mathcal{L}$ на обучающем наборе, вычислении градиентов оценки потери по параметрам Θ и сдвиге параметров в направлении, противоположном градиенту. Различные методы оптимизации отличаются способом вычисления оценки ошибки и определением «сдвига в направлении, противоположном градиенту». Мы опишем базовый алгоритм *стохастического градиентного спуска* (СГС), а затем кратко упомянем другие подходы и дадим ссылки на литературу.

[1] Lasso в данном случае – аббревиатура *least absolute shrinkage and selection operator* и ни малейшего отношения к аркану не имеет, поэтому термин оставлен без перевода. – *Прим. перев.*

ОБОСНОВАНИЕ ГРАДИЕНТНОЙ ОПТИМИЗАЦИИ Рассмотрим задачу о нахождении скалярного значения x, доставляющего минимум функции $y = f(x)$. Канонический подход – вычислить производную $f'(x)$ и решить уравнение $f'(x) = 0$ для нахождения точек экстремума. Но предположим, что этим подходом воспользоваться нельзя (это и вправду проблематично для функции многих переменных). Альтернативный подход численный: вычислить производную $f'(x)$ и выбрать начальное приближение x_i. Вычисление $u = f'(x_i)$ даст нам направление изменения. Если $u = 0$, то x_i – точка экстремума. В противном случае сдвинуться в противоположном u направлении, положив $x_{i+1} \leftarrow x_i - \eta u$, где η – *параметр скорости*. При достаточно малом значении η величина $f(x_{i+1})$ будет меньше $f(x_i)$. Повторяя этот процесс (с убывающими значениями η), мы найдем точку экстремума x_i. Если функция $f()$ выпуклая, то экстремум будет глобальным, в противном случае гарантируется только нахождение локального экстремума.

Градиентная оптимизация просто обобщает эту идею на функции нескольких переменных. Градиентом функции k переменных называется набор k частных производных по каждой переменной. Сдвиг входных данных в направлении градиента увеличивает значение функции, а сдвиг в противоположном направлении – уменьшает. При оптимизации потери $\mathcal{L}(\Theta; \boldsymbol{x}_{1:n}, \mathbf{y}_{1:n})$ параметры Θ рассматриваются как входные данные для функции, а обучающие примеры – как константы.

ВЫПУКЛОСТЬ При выполнении градиентной оптимизации принято различать *выпуклые* (или *вогнутые*) и *невыпуклые* (*невогнутые*) функции. *Выпуклой* называется функция, вторая производная которой всюду неотрицательна[1]. Следовательно, у выпуклой функции может быть только одна точка минимума. Аналогично *вогнутой* называется функция, вторая производная которой всюду неположительна, поэтому она имеет единственную точку максимума. Выпуклые (вогнутые) функции обладают тем свойством, что их легко минимизировать (максимизировать) методом градиентной оптимизации – нужно просто следовать в направлении градиента, пока не будет достигнута точка экстремума. Найдя ее, мы можем быть уверены, что этот экстремум глобальный. Напротив, если функция не является ни выпуклой, ни вогнутой, то метод градиентной оптимизации может сойтись к локальному экстремуму, не заметив глобальный.

2.8.1. Стохастический градиентный спуск

Эффективный метод обучения линейных моделей дает алгоритм СГС [Bottou, 2012, LeCun et al., 1998a] и его варианты. СГС – это общий алгоритм оптимизации. Он принимает функцию f с параметрами Θ, функцию потерь L и пары вход-выход,

[1] Выпуклая функция не обязана быть дважды дифференцируемой, поэтому определение неверно. К тому же выпуклость – локальное свойство, так что слово «всюду» неуместно. Оставим эту нестрогость на совести автора. – *Прим. перев.*

$\boldsymbol{x}_{1:n}, \boldsymbol{y}_{1:n}$. Затем он пытается найти такие параметры Θ, чтобы суммарная потеря f на обучающих примерах была мала. Работа СГС показана в алгоритме 2.1.

Цель алгоритма – установить параметры Θ таким образом, чтобы минимизировать полную потерю $\mathcal{L}(\Theta) = \sum_{i=1}^{n} L(f(\boldsymbol{x}_i; \theta), \boldsymbol{y}_i)$ на обучающем наборе. Для этого он в цикле выбирает очередной пример и вычисляет на нем градиент ошибки по параметрам Θ (строка 4) – вход и ожидаемый выход предполагаются фиксированными, а потеря рассматривается как функция параметров Θ. Параметры Θ затем обновляются в направлении, противоположном градиенту, и умножаются на скорость обучения η_t (строка 5). Скорость обучения можно либо зафиксировать на протяжении всего процесса обучения, либо уменьшать как функцию от номера шага t[1]. Более подробное обсуждение скорости обучения *см*. в разделе 5.2.

Алгоритм 2.1. Онлайновое обучение методом стохастического градиентного спуска

Вход:
– Функция $f(\boldsymbol{x}; \Theta)$ с параметрами Θ
– Обучающий набор входов $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ и желаемых выходов $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$
– Функция потерь L

1: **while** критерий останова не выполнен **do**
2: выбрать обучающий пример $\boldsymbol{x}_i, \boldsymbol{y}_i$
3: вычислить потерю $L(f(\boldsymbol{x}_i; \Theta), \boldsymbol{y}_i)$
4: $\hat{\boldsymbol{g}} \leftarrow$ градиенты $L(f(\boldsymbol{x}_i; \Theta), \boldsymbol{y}_i)$ по Θ
5: $\Theta \leftarrow \Theta - \eta_t \hat{\boldsymbol{g}}$
6: **return** Θ

Заметим, что ошибка, вычисленная в строке 3, основана на одном обучающем примере и потому является лишь грубой оценкой потери $\mathcal{L}$ на всем корпусе, которую мы стремимся минимизировать. Шум при вычислении потери может привести к неточным градиентам. Стандартный способ уменьшения шума – оценивать оценку и градиенты на основе выборки, содержащей m примеров. Так мы получаем *мини-пакетный алгоритм СГС* (алгоритм 2.2).

В строках 3–6 алгоритм оценивает градиент полной потери на основе мини-пакета. По завершении цикла $\hat{\boldsymbol{g}}$ содержит оценку градиента, а параметры Θ обновлены в направлении $\hat{\boldsymbol{g}}$. Размер мини-пакета может изменяться от $m = 1$ до $m = n$. Чем больше мини-пакет, тем лучше оценка градиентов. С другой стороны, меньшие пакеты дают возможность выполнить больше обновлений и, значит, ускорить сходимость. Помимо более точной оценки градиентов, мини-пакетный алгоритм открывает возможности для повышения эффективности обучения. При небольших значениях m некоторые компьютерные архитектуры (конкретно GPU) допускают эффективное распараллеливание вычисления в строках 3–6. При правильном выборе убывающей скорости обучения гарантируется, что СГС

[1] Уменьшать скорость обучения необходимо, для того чтобы можно было доказать сходимость СГС.

сходится к глобальному оптимуму, если функция выпукла. Именно так обстоит дело в линейных и логлинейных моделях в сочетании с функциями потерь и регуляризаторами, описанными в этой главе. Однако СГС можно использовать и для оптимизации невыпуклых функций, в том числе многослойных нейронных сетей. Хотя в таком случае нахождение глобального оптимума уже не гарантируется, на практике алгоритм ведет себя хорошо и считается надежным[1].

Алгоритм 2.2. Обучение методом мини-пакетного стохастического градиентного спуска

Вход:
– Функция $f(\boldsymbol{x}; \Theta)$ с параметрами Θ
– Обучающий набор входов $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ и желаемых выходов $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$
– Функция потерь L

while критерий останова не выполнен **do**
  выбрать мини-пакет, содержащий m примеров $\{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_m, \boldsymbol{y}_m)\}$
  $\hat{\boldsymbol{g}} \leftarrow 0$
  for $i = 1$ to m **do**
    вычислить потерю $L(f(\boldsymbol{x}_i; \Theta), \boldsymbol{y}_i)$
    $\hat{\boldsymbol{g}} \leftarrow \hat{\boldsymbol{g}} +$ градиенты $(1/m)\, L(f(\boldsymbol{x}_i; \Theta), \boldsymbol{y}_i)$ по Θ
  $\Theta \leftarrow \Theta - \eta_t \hat{\boldsymbol{g}}$
return Θ

2.8.2. Полный пример

В качестве примера рассмотрим многоклассовый линейный классификатор с кусочно-линейной функцией потерь:

$$\begin{aligned}
\hat{y} &= \operatorname*{argmax}_i \hat{\boldsymbol{y}}_{[i]};\\
\hat{\boldsymbol{y}} &= f(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{W} + \boldsymbol{b};\\
L(\hat{\boldsymbol{y}}, \boldsymbol{y}) &= \max(0, 1 - (\hat{\boldsymbol{y}}_{[t]} - \hat{\boldsymbol{y}}_{[k]}))\\
&= \max(0, 1 - ((\boldsymbol{x}\boldsymbol{W} + \boldsymbol{b})_{[t]} - (\boldsymbol{x}\boldsymbol{W} + \boldsymbol{b})_{[k]}));\\
t &= \operatorname*{argmax}_i \boldsymbol{y}_{[i]};\\
k &= \operatorname*{argmax}_i \hat{\boldsymbol{y}}_{[i]}, i \neq t.
\end{aligned}$$

Мы хотим установить параметры $\boldsymbol{W}$ и $\boldsymbol{b}$ так, чтобы потеря достигала минимума. Требуется вычислить градиенты потери по значениям $\boldsymbol{W}$ и $\boldsymbol{b}$. Градиент – это набор частных производных по каждой переменной:

[1] В недавних работах по нейронным сетям утверждается, что невыпуклость сети проявляется в появлении многочисленных седловых точек вместо локальных минимумов [Dauphin et al., 2014]. Этим может объясняться успех обучения нейронных сетей, несмотря на применение локальных методов поиска.

$$\frac{\partial L(\hat{\mathbf{y}},\mathbf{y})}{\partial \boldsymbol{W}}=\begin{pmatrix}\frac{\partial L(\hat{\mathbf{y}},\mathbf{y})}{\partial \boldsymbol{W}_{[1,1]}} & \frac{\partial L(\hat{\mathbf{y}},\mathbf{y})}{\partial \boldsymbol{W}_{[1,2]}} & \cdots & \frac{\partial L(\hat{\mathbf{y}},\mathbf{y})}{\partial \boldsymbol{W}_{[1,n]}}\\ \frac{\partial L(\hat{\mathbf{y}},\mathbf{y})}{\partial \boldsymbol{W}_{[2,1]}} & \frac{\partial L(\hat{\mathbf{y}},\mathbf{y})}{\partial \boldsymbol{W}_{[2,2]}} & \cdots & \frac{\partial L(\hat{\mathbf{y}},\mathbf{y})}{\partial \boldsymbol{W}_{[2,n]}}\\ \vdots & \vdots & \ddots & \vdots\\ \frac{\partial L(\hat{\mathbf{y}},\mathbf{y})}{\partial \boldsymbol{W}_{[m,1]}} & \frac{\partial L(\hat{\mathbf{y}},\mathbf{y})}{\partial \boldsymbol{W}_{[m,2]}} & \cdots & \frac{\partial L(\hat{\mathbf{y}},\mathbf{y})}{\partial \boldsymbol{W}_{[m,n]}}\end{pmatrix};$$

$$\frac{\partial L(\hat{\mathbf{y}},\mathbf{y})}{\partial \boldsymbol{b}}=\begin{pmatrix}\frac{\partial L(\hat{\mathbf{y}},\mathbf{y})}{\partial \boldsymbol{b}_{[1]}} & \frac{\partial L(\hat{\mathbf{y}},\mathbf{y})}{\partial \boldsymbol{b}_{[2]}} & \cdots & \frac{\partial L(\hat{\mathbf{y}},\mathbf{y})}{\partial \boldsymbol{b}_{[n]}}\end{pmatrix}.$$

Конкретно мы вычисляем производные потери по каждому значению $\boldsymbol{W}_{[i,j]}$ и $\boldsymbol{b}_{[j]}$. Начнем с раскрытия членов в функции потерь[1]:

$$\begin{aligned}L(\hat{\mathbf{y}},\mathbf{y}) &= \max(0,1-(\hat{\mathbf{y}}_{[t]}-\hat{\mathbf{y}}_{[k]}))\\ &= \max(0,1-((\boldsymbol{x}\boldsymbol{W}+\boldsymbol{b})_{[t]}-(\boldsymbol{x}\boldsymbol{W}+\boldsymbol{b})_{[k]}));\\ &= \max\left(0,1-\left(\left(\sum_i \boldsymbol{x}_{[i]}\cdot\boldsymbol{W}_{[i,t]}+\boldsymbol{b}_{[t]}\right)-\left(\sum_i \boldsymbol{x}_{[i]}\cdot\boldsymbol{W}_{[i,k]}+\boldsymbol{b}_{[k]}\right)\right)\right)\\ &= \max\left(0,1-\sum_i \boldsymbol{x}_{[i]}\cdot\boldsymbol{W}_{[i,t]}+\boldsymbol{b}_{[t]}+\sum_i \boldsymbol{x}_{[i]}\cdot\boldsymbol{W}_{[i,k]}+\boldsymbol{b}_{[k]}\right);\\ t &= \underset{i}{\operatorname{argmax}}\,\mathbf{y}_{[i]};\\ k &= \underset{i}{\operatorname{argmax}}\,\hat{\mathbf{y}}_{[i]},\ i\neq t.\end{aligned}$$

Прежде всего заметим, что если $1-(\hat{\mathbf{y}}_{[t]}-\hat{\mathbf{y}}_{[k]})\le 0$, то потеря равна 0, а вместе с ней и градиент (производная функции max – это производная максимального значения). В противном случае рассмотрим производную $\partial L/\partial \boldsymbol{b}_{[i]}$. В этой частной производной $\boldsymbol{b}_{[i]}$ рассматривается как переменная, а все остальные параметры – как константы. Для $i\neq k, t$ член $\boldsymbol{b}_{[i]}$ не дает вклада в потерю, и его производная равна 0. Для $i=k$ и $i=t$ имеем:

$$\frac{\partial L}{\partial \boldsymbol{b}_{[i]}}=\begin{cases}-1 & i=t\\ 1 & i=k\\ 0 & \text{в противном случае}\end{cases}.$$

Аналогично для производных по $\boldsymbol{W}_{[i,j]}$ вклад в потерю ненулевой, только если $j=k$ или $j=t$. Имеем:

[1] Более продвинутая техника вычислений позволяет работать с матрицами и векторами напрямую. Но здесь мы придерживаемся школьных методов.

$$\frac{\partial L}{\partial \boldsymbol{W}_{[i,j]}} = \begin{cases} \dfrac{\partial(-\boldsymbol{x}_{[i]} \cdot \boldsymbol{W}_{[i,t]})}{\partial \boldsymbol{W}_{[i,t]}} & = -\boldsymbol{x}_{[i]} \quad j = t \\ \dfrac{\partial(\boldsymbol{x}_{[i]} \cdot \boldsymbol{W}_{[i,k]})}{\partial \boldsymbol{W}_{[i,k]}} & = \boldsymbol{x}_{[i]} \quad j = k \\ 0 & \text{в противном случае} \end{cases}.$$

На этом вычисление градиента завершается.

Читателю предлагается выполнить простое упражнение: вычислить градиенты многоклассовой линейной модели с кусочно-линейной функцией потерь и L_2-регуляризацией, а также градиенты многоклассовой классификации с преобразованием выхода softmax и перекрестной энтропией в качестве потери.

2.8.3. Не только СГС

Хотя алгоритм СГС может давать и часто дает хорошие результаты, имеются и более передовые алгоритмы. Алгоритмы *СГС+импульс* [Polyak, 1964] и *импульс Нестерова* [Nesterov, 1983, 2004, Sutskever et al., 2013] представляют собой варианты СГС, в которых ранее вычисленные градиенты аккумулируются и оказывают влияние на текущее обновление. Алгоритмы с адаптивной скоростью обучения, в том числе AdaGrad [Duchi et al., 2011], AdaDelta [Zeiler, 2012], RMSProp [Tieleman and Hinton, 2012] и Adam [Kingma and Ba, 2014], автоматически выбирают скорость обучения для каждого мини-пакета, иногда по каждой координате в отдельности, что потенциально устраняет необходимость в трудоемком подборе этого гиперпараметра. Детальное описание этих алгоритмов *см*. в оригинальных статьях или в книге [Bengio et al., 2016, разделы 8.3, 8.4][1].

[1] *Иошуа Бенджио, Ян Гудфеллоу, Аарон Курвилль*. Глубокое обучение. М.: ДМК Пресс, 2017.

Глава 3

От линейных моделей к многослойным перцептронам

3.1. Ограничения линейных моделей: проблема XOR

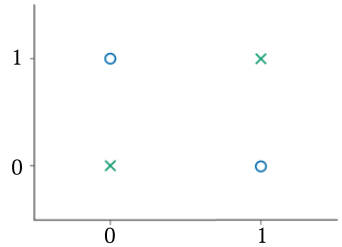
Класс гипотез, состоящий из линейных (и логлинейных) моделей, сильно ограничен. Например, он не позволяет представить функцию XOR, определенную следующим образом:

$$\begin{aligned}&\text{xor}(0,0)=0;\\&\text{xor}(1,0)=1;\\&\text{xor}(0,1)=1;\\&\text{xor}(1,1)=0.\end{aligned}$$

Это означает, что не существует такой параметризации $\boldsymbol{w}\in\mathbb{R}^2$, $b\in\mathbb{R}$, что:

$$\begin{aligned}&(0,0)\cdot\boldsymbol{w}+b<0;\\&(0,1)\cdot\boldsymbol{w}+b\geq 0;\\&(1,0)\cdot\boldsymbol{w}+b\geq 0;\\&(1,1)\cdot\boldsymbol{w}+b<0.\end{aligned}$$

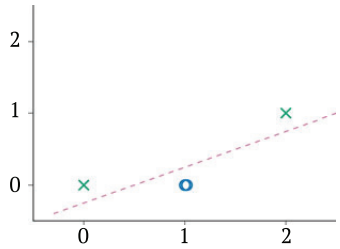
Чтобы понять, почему это так, рассмотрим следующий график функции XOR, на котором синими кружочками обозначен положительный класс, а зелеными крестиками – отрицательный.



Очевидно, что не существует прямой, разделяющей эти два класса.

3.2. Нелинейные преобразования входа

Однако если преобразовать точки с помощью нелинейной функции $\phi(x_1, x_2) = [x_1 \times x_2, x_1 + x_2]$, то проблема XOR становится линейно разделимой:



Функция ϕ преобразовала данные в представление, пригодное для линейной классификации. Имея в своем распоряжении ϕ, мы теперь можем легко обучить линейный классификатор для решения проблемы XOR:

$$\hat{\boldsymbol{y}} = f(\boldsymbol{x}) = \phi(\boldsymbol{x})\boldsymbol{W} + \boldsymbol{b}.$$

В общем случае мы можем обучить линейный классификатор на линейно неразделимом наборе данных, определив функцию, отображающую данные в такое представление, которое допускает линейное разделение, а затем обучив линейный классификатор на получившемся представлении. В примере XOR преобразованные данные были такой же размерности, как исходные, но часто для получения линейно разделимых данных приходится переходить в пространство гораздо более высокой размерности.

Но у этого решения имеется бросающийся в глаза недостаток: мы должны вручную определить функцию ϕ, зависящую от конкретного набора данных, и этот процесс требует от человека изрядной изобретательности.

3.3. Ядерные методы

Ядерный метод опорных векторов (SVM) [Boser and et al., 1992] и ядерные методы в общем случае [Shawe-Taylor and Cristianini, 2004] знаменуют подход к проблеме, при котором определяется набор общих отображений, каждое из которых переводит данные в пространство очень высокой, иногда даже бесконечной, размерности, а затем производится линейная классификация в преобразованном пространстве. Работа в пространствах очень высокой размерности существенно повышает шансы найти подходящий линейный разделитель.

Примером может служить *полиномиальное отображение* $\phi(\boldsymbol{x}) = (\boldsymbol{x})^d$. При $d = 2$ получаем $\phi(x_1, x_2) = (x_1x_1, x_1x_2, x_2x_1, x_2x_2)$. Это дает нам все комбинации двух переменных, что позволяет решить проблему XOR с помощью линейного классификатора ценой полиномиального увеличения количества параметров. В проблеме XOR это отображение повысило размерность входа (а значит, и количество параметров) с 2 до 4. В примере определения языка размерность входа увеличилась бы с 784 до $784^2 = 614\,656$.

Для работы в пространстве очень высокой размерности может не хватить вычислительных ресурсов, поэтому в ядерных методах применяется изобретатель-

ный *ядерный трюк* [Aizerman et al., 1964, Schölkopf, 2001], который позволяет работать в преобразованном пространстве, вообще не вычисляя преобразованное представление. Предложены отображения общего вида для работы во многих типичных ситуациях, а пользователю остается выбрать то, которое подходит для его задачи (зачастую методом проб и ошибок). Недостаток такого подхода заключается в том, что применение ядерного трюка делает процедуру классификации для SVM линейно зависящей от размера обучающего набора, так что использовать ее для больших обучающих наборов оказывается невозможно. Еще один недостаток многомерных пространств – повышенный риск переобучения.

3.4. Обучаемые отображающие функции

Другой подход состоит в том, чтобы определить *обучаемую* нелинейную отображающую функцию и обучить ее вместе с линейным классификатором. То есть поиск подходящего представления возлагается на алгоритм обучения. Например, отображающая функция может принимать вид параметрической линейной модели, сопровождаемой нелинейной функцией активации g, которая применяется к каждой размерности выхода:

$$\begin{aligned} \hat{\boldsymbol{y}} &= \phi(\boldsymbol{x})\boldsymbol{W} + \boldsymbol{b} \\ \phi(\boldsymbol{x}) &= g(\boldsymbol{x}\boldsymbol{W}' + \boldsymbol{b}'). \end{aligned} \tag{3.1}$$

Положив $g(x) = \max(0, x)$ и $\boldsymbol{W}' = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \boldsymbol{b}' = (-1, 0)$, мы получим отображение, эквивалентное $(x_1 \times x_2, x_1 + x_2)$ для интересующих нас точек (0,0), (0,1), (1,0), (1,1). Это отображение решает проблему XOR. Полная функция $g(\boldsymbol{x}\boldsymbol{W}' + \boldsymbol{b}')\boldsymbol{W} + \boldsymbol{b}$ дифференцируемая (хотя и не выпуклая), что позволяет применить к обучению модели градиентные методы. В результате будут одновременно обучены функция представления и надстроенный над ней линейный классификатор. В этом и состоит основная идея глубокого обучения и нейронных сетей. На самом деле уравнение (3.1) описывает очень распространенную архитектуру нейронной сети – *многослойный перцептрон* (МСП). Установив, чего мы хотим достичь, можем обратиться к более подробному описанию многослойных нейронных сетей.

Глава 4

Нейронные сети прямого распространения

4.1. Метафора, инспирированная мозгом

Как следует из самого названия, нейронные сети были инспирированы механизмом вычислений в мозге, который состоит из вычислительных блоков – нейронов. Хотя аналогия между искусственными нейронными сетями и мозгом на самом деле довольно шаткая, мы для полноты картины все же приведем эту метафору. Нейрон в ней рассматривается как вычислительный блок, имеющий входы и выходы. С каждым входом ассоциирован вес. Нейрон умножает каждый вход на его вес, затем вычисляет сумму[1], применяет к результату нелинейную функцию и передает ее значение на свой выход. На рис. 4.1 показан такой нейрон.

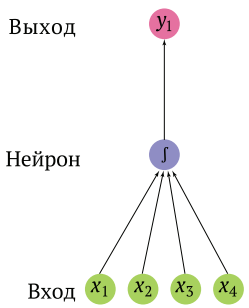


Рис. 4.1 ❖ Один нейрон с четырьмя входами

Нейроны связаны друг с другом и образую сеть: выход нейрона может подаваться на входы одного или более нейронов. Показано, что такие сети являются весьма эффективными вычислительными устройствами. Если веса подобраны правильно, то нейронная сеть с достаточно большим количеством нейронов и нелинейной функцией активации может аппроксимировать очень широкий круг математических функций (ниже мы уточним эту деталь).

---

[1] Суммирование – самая распространенная операция, но возможны и другие, например max.

Типичную сеть прямого распространения можно изобразить, как показано на рис. 4.2. Каждый кружочек – это нейрон, входящие стрелки – входы нейрона, исходящие стрелки – его выходы. С каждой стрелкой ассоциирован вес, отражающий ее важность (не показаны). Нейроны организованы в слои, отражающие поток информации. Для нейронов нижнего слоя нет входящих стрелок, а для нейронов верхнего слоя – исходящих; они образуют выход сети. Остальные слои называются «скрытыми». Значок сигмообразной формы внутри нейронов средних слоев представляет нелинейную функцию (например, логистическую функцию $1/(1 + e^{-x})$), которая применяется к значению нейрона до передачи его на выход. На этом рисунке каждый нейрон соединен со всеми нейронами следующего слоя – такой слой называется *полносвязным*, или *аффинным*.

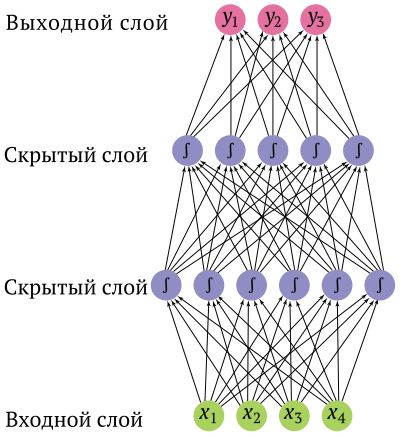


Рис. 4.2 ❖ Нейронная сеть прямого распространения с двумя скрытыми слоями

Хотя метафора мозга красивая и интригующая, она также отвлекает от сути дела и неудобна для математических операций. Поэтому вернемся к более краткой математической нотации. Как станет ясно очень скоро, сеть прямого распространения типа показанной на рис. 4.2 – это просто стопка линейных моделей, проложенная нелинейными функциями. Значения нейронов в каждом ряду сети можно рассматривать как вектор. На рис. 4.2 входным слоем является четырехмерный вектор ($\boldsymbol{x}$), а расположенный над ним слой – шестимерный вектор ($\boldsymbol{h}^1$). Полносвязный слой можно интерпретировать как линейное преобразование из четырехмерного пространства в шестимерное. Полносвязный слой реализует умножение вектора на матрицу, $\boldsymbol{h} = \boldsymbol{xW}$, где вес связи между i-м нейроном входной строки и j-м нейроном выходной строки равен $\boldsymbol{W}_{[i,j]}$[1]. Значения $\boldsymbol{h}$ затем преобразуются нелинейной функцией g, которая применяется к каждому значению перед его передачей на следующий уровень. Все вычисление от входа до выхода можно записать в виде $(\boldsymbol{g}(\boldsymbol{xW}^1))\boldsymbol{W}^2$, где $\boldsymbol{W}^1$ – веса первого слоя, а $\boldsymbol{W}^2$ – веса второго слоя. Если принять такую точку зрения, то единственный нейрон на рис. 4.1

[1] Чтобы понять, почему это так, обозначим $\boldsymbol{W}_{[i,j]}$ вес i-го входа j-го нейрона $\boldsymbol{h}$. Значение $\boldsymbol{h}_{[j]}$ тогда будет равно $\boldsymbol{h}_{[j]} = \sum_{i=1}^{4} \boldsymbol{x}_{[i]} \cdot \boldsymbol{W}_{[i,j]}$.

эквивалентен логистическому (логлинейному) бинарному классификатору $\sigma(\boldsymbol{xw})$ без члена смещения.

4.2. Математическая нотация

Начиная с этого момента, мы оставим метафору мозга в покое, а будем описывать сети исключительно в терминах операций над векторами и матрицами.

Простейшая нейронная сеть называется *перцептроном*. Это просто линейная модель:

$$\begin{gathered}\mathrm{NN}_{\mathrm{Perceptron}}(\boldsymbol{x}) = \boldsymbol{xW} + \boldsymbol{b}, \\ \boldsymbol{x} \in \mathbb{R}^{d_{in}}, \quad \boldsymbol{W} \in \mathbb{R}^{d_{in} \times d_{out}}, \quad \boldsymbol{b} \in \mathbb{R}^{d_{out}},\end{gathered} \tag{4.1}$$

где $\boldsymbol{W}$ – матрица весов, а $\boldsymbol{b}$ – смещение[1]. Чтобы выйти за рамки линейных функций, мы вводим нелинейный скрытый слой (в сети на рис. 4.2 таких слоев два), так что получается многослойный линейный перцептрон с одним скрытым слоем (МСП1). Нейронная сеть прямого распространения с одним скрытым слоем имеет вид:

$$\begin{gathered}\mathrm{NN}_{\mathrm{MLP1}}(\boldsymbol{x}) = \mathrm{g}(\boldsymbol{xW^1} + \boldsymbol{b^1})\boldsymbol{W^2} + \boldsymbol{b^2}. \\ \boldsymbol{x} \in \mathbb{R}^{d_{in}}, \quad \boldsymbol{W^1} \in \mathbb{R}^{d_{in} \times d_1}, \quad \boldsymbol{b^1} \in \mathbb{R}^{d_1}, \quad \boldsymbol{W^2} \in \mathbb{R}^{d_1 \times d_2}, \quad \boldsymbol{b^2} \in \mathbb{R}^{d_2}.\end{gathered} \tag{4.2}$$

Здесь $\boldsymbol{W^1}$ и $\boldsymbol{b^1}$ – матрица и смещение первого линейного преобразования входа, g – нелинейная функция, применяемая к каждому элементу (она называется *нелинейностью*, или *функцией активации*), а $\boldsymbol{W^2}$ и $\boldsymbol{b^2}$ – матрица и смещение второго линейного преобразования.

Итак, $\boldsymbol{xW^1} + \boldsymbol{b^1}$ – линейное преобразование входа из d_{in}-мерного пространства в d_1-мерное. Затем к каждому из d_1-измерений применяется функция g, а матрица $\boldsymbol{W^2}$ в сочетании со смещением $\boldsymbol{b^2}$ используется для преобразования результата в d^2-мерный выходной вектор. Нелинейная функция активации g играет главную роль в способности сети представлять сложные функции. Не будь этой нелинейности, нейронная сеть могла бы представлять только линейные преобразования входа[2]. В терминах главы 3 первый слой преобразует данные в подходящее представление, а второй применяет к этому представлению линейный классификатор.

Мы можем также добавить еще одно линейное преобразование и нелинейность, получив тем самым МСП с двумя скрытыми уровнями (именно так устроена сеть на рис. 4.2):

$$\mathrm{NN}_{\mathrm{MLP2}}(\boldsymbol{x}) = (g^2(g^1(\boldsymbol{xW^1} + \boldsymbol{b^1})\boldsymbol{W^2} + \boldsymbol{b^2})\boldsymbol{W}. \tag{4.3}$$

Пожалуй, запись более глубоких сетей будет понятнее, если использовать промежуточные переменные:

---

[1] В сети на рис. 4.2 нет смещения. Его можно добавить в слой, включив дополнительный нейрон без входящих связей, который всегда возвращает значение 1.

[2] Чтобы убедиться в этом, примите во внимание, что последовательность линейных преобразований сама является линейным преобразованием.

$$\begin{aligned}\mathrm{NN}_{\mathrm{MLP2}}(\boldsymbol{x}) &= \boldsymbol{y};\\ \boldsymbol{h}^1 &= g^1(\boldsymbol{x}\boldsymbol{W}^1 + \boldsymbol{b}^1);\\ \boldsymbol{h}^2 &= g^2(\boldsymbol{h}^1\boldsymbol{W}^2 + \boldsymbol{b}^2);\\ \boldsymbol{y} &= \boldsymbol{h}^2\boldsymbol{W}^3.\end{aligned} \tag{4.4}$$

Вектор, получающийся в результате каждого линейного преобразования, называется *слоем*. Самое внешнее линейное преобразование порождает *выходной слой*, а остальные – *скрытые слои*. За каждым скрытым слоем следует нелинейная активация. В некоторых случаях, в частности в последнем слое нашего примера, вектор смещений обнуляется.

Слои, являющиеся результатами линейных преобразований, часто называют *полносвязными*, или *аффинными*. Существуют и другие архитектуры. В частности, в задачах распознавания образов хорошие результаты дают *сверточные* и *пулинговые* слои. Такие слои используются и в лингвистической обработке и будут обсуждаться в главе 13. Сети с несколькими скрытыми слоями называются *глубокими*, отсюда и название *глубокое обучение*.

При описании нейронной сети следует задавать *размерности* слоев и входных данных. Слой ожидает получить на входе d_{in}-мерный вектор и выводит d_{out}-мерный вектор. Размерностью слоя считаются размерности его выхода. Для полносвязного слоя $l(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}$ с размерностью входа d_{in} и размерностью выхода d_{out} размерность $\boldsymbol{x}$ равна $1 \times d_{in}$, размерность $\boldsymbol{W}$ – $d_{in} \times d_{out}$ и размерность $\boldsymbol{b}$ – $1 \times d_{out}$.

Как и в случае линейных моделей, выходом нейронной сети является d_{out}-мерный вектор. Если $d_{out} = 1$, то выход сети называется скалярным. Такие сети можно использовать для регрессии (или балльного оценивания), если учитывать выходное значение целиком, или для бинарной классификации, если учитывать только его знак. Сети, для которых $d_{out} = k > 1$, можно использовать для k-классовой классификации, если ассоциировать каждое измерение с классом и искать измерение с максимальным значением. Кроме того, если все элементы выходного вектора неотрицательны и в сумме составляют единицу, то выход можно интерпретировать как распределение вероятностей классов (для такой нормировки выхода обычно применяют функцию softmax к выходному слою (*см.* раздел 2.6)).

Матрицы и смещения, определяющие линейные преобразования, являются *параметрами* сети. Как и в линейных моделях, набор всех параметров принято обозначать Θ. В сочетании с входом параметры определяют выход сети. Алгоритм обучения призван выбрать такие значения параметров, чтобы предсказания сети были правильны. В отличие от линейных моделей, функция потерь многослойной нейронной сети относительно параметров не является выпуклой[1], поэтому для поиска оптимальных значений параметров не хватит никаких вычислительных ресурсов. Тем не менее градиентные методы оптимизации, описанные в разделе 2.8, применимы и на практике показывают отличные результаты. Обучение нейронных сетей подробно обсуждается в главе 5.

[1] Для строго выпуклых функций существует только одно оптимальное решение, поэтому их легко оптимизировать градиентными методами.

4.3. Репрезентативная способность

В работах Hornik et al. [1989] и Cybenko [1989] показано, что МСП1 является универсальным аппроксиматором, т. е. может аппроксимировать с произвольной ненулевой точностью семейство функций, включающее все непрерывные функции на замкнутом ограниченном подмножестве $\mathbb{R}^n$, а также любую функцию, отображающую произвольное конечномерное дискретное пространство в другое подобное[1]. Отсюда можно было бы сделать вывод, что нет причин рассматривать более сложные архитектуры, чем МСП1. Однако указанный теоретический результат ничего не говорит об обучаемости нейронной сети (утверждается лишь, что представление существует, но вопрос о том, насколько легко или трудно установить параметры для конкретных обучающих данных и алгоритма обучения, обойден молчанием). Не гарантируется также, что алгоритм обучения находит *правильную* функцию, генерирующую обучающие данные. Наконец, ни слова не сказано о том, насколько большим должен быть скрытый слой. В работе Telgarsky [2016] показано, что существуют нейронные сети с большим числом слоев ограниченного размера, которые невозможно аппроксимировать сетями с меньшим числом слоев, если только эти слои не будут экспоненциально большими.

На практике мы обучаем нейронные сети на относительно небольших объемах данных, применяя локальные методы поиска, например варианты стохастического градиентного спуска, и используем скрытые слои относительно скромного размера (до нескольких тысяч нейронов). Поскольку теорема об универсальной аппроксимации не дает никаких гарантий при таких неидеальных условиях, определенно имеет смысл попробовать более сложные архитектуры, чем МСП1. Однако во многих случаях МСП1 все-таки дает хорошие результаты. Дополнительные сведения о репрезентативной способности нейронных сетей прямого распространения *см*. в книге Bengio et al. [2016, раздел 6.5].

4.4. Стандартные нелинейности

Нелинейность g может принимать различные формы. В настоящее время не существует хорошей теории, которая подсказывала бы, какую нелинейность в каких условиях применять, поэтому выбор подходящей для данной задачи нелинейности обычно осуществляется эмпирически. Ниже мы рассмотрим типичные нелинейности, встречающиеся в литературе: сигмоида, tanh, hardtanh и блок линейной ректификации (rectified linear unit – ReLU). Некоторые исследователи NLP экспериментировали с другими нелинейностями, например кубической функцией и tanh в кубе.

СИГМОИДА Сигмоидная функция активации $\sigma(x) = 1/(1 + e^{-x})$, называемая также логистической функцией, имеет S-образную форму и преобразует любое значение x в число из диапазона [0, 1]. Сигмоида была канонической нелинейностью

[1] Точнее говоря, сеть прямого распространения с линейным выходным слоем и хотя бы одним скрытым слоем со «сплющивающей» функцией активации может аппроксимировать произвольную измеримую по Борелю функцию из одного конечномерного пространства в другое. Впоследствии доказательство было обобщено в работе Leshno et al. [1993] на более широкое множество функций активации, включающее блоки ReLU вида $g(x) = \max(0, x)$.

с самого зарождения нейронных сетей, но в настоящее время считается неподходящей для использования во внутренних слоях сети, поскольку описанные ниже альтернативы демонстрируют гораздо более высокое эмпирическое качество.

ГИПЕРБОЛИЧЕСКИЙ ТАНГЕНС (TANH) Функция активации $\tanh(x) = (e^{2x} - 1)/(e^{2x} + 1)$ тоже имеет S-образную форму и преобразует любое x в число из диапазона $[-1, 1]$.

HARDTANH Функция hardtanh является аппроксимацией tanh, но она сама и ее производные вычисляются быстрее:

$$\text{hardtanh}(x) = \begin{cases} -1 & x < -1 \\ 1 & x > 1 \\ x & \text{в противном случае} \end{cases}. \tag{4.5}$$

БЛОК ЛИНЕЙНОЙ РЕКТИФИКАЦИИ (RELU) Это очень простая функция активации [Glorot et al., 2011], с которой легко работать. Во многих работах показано, что она дает отличные результаты[1]. Блок ReLU заменяет все значения $x < 0$ нулями. Несмотря на свою простоту, он прекрасно работает во многих задачах, особенно в сочетании с регуляризирующей техникой прореживания (*см.* раздел 4.6):

$$\text{ReLU}(x) = \max(0, x) = \begin{cases} 0 & x < 0 \\ x & \text{в противном случае} \end{cases}. \tag{4.6}$$

В общем и целом блоки типа ReLU и tanh работают хорошо и значительно превосходят сигмоиду. Имеет смысл поэкспериментировать с ними, поскольку одна функция может оказаться лучше в одних ситуациях, а другая – в других.

На рис. 4.3 показаны формы различных функций активации и их производных.

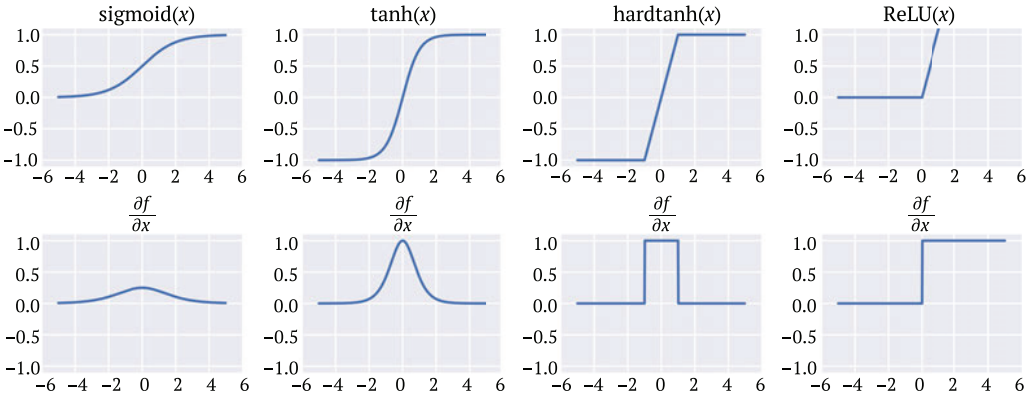


Рис. 4.3 ❖ Функции активации (вверху) и их производные (внизу)

[1] Технически функция активации ReLU превосходит сигмоиду и tanh, потому что не содержит вычислительно сложных функций и, главное, потому что не подвержена насыщению. Сигмоида и tanh асимптотически приближаются к 1, и градиенты в этой области близки к нулю, в результате чего и полный градиент близок к нулю. У функции активации ReLU этой проблемы нет, поэтому она особенно хороша для многослойных сетей, чувствительных к проблеме исчезающего градиента при обучении с насыщающимися блоками.

4.5. Функции потерь

При обучении нейронной сети (подробнее об этом *см*. главу 5), как и при обучении линейного классификатора, задается функция потерь $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$, определяющая потерю в случае, когда предсказано значение $\hat{\boldsymbol{y}}$, хотя истинным является значение $\boldsymbol{y}$. Цель обучения – минимизировать полную потерю на всех обучающих примерах. Потеря $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ сопоставляет числовую оценку (скаляр) выходу сети $\hat{\boldsymbol{y}}$ при условии, что истинный ожидаемый выход равен $\boldsymbol{y}$. Функции потерь для линейных моделей, рассмотренные в разделе 2.7.1, актуальны и широко используются также в нейронных сетях. Подробное обсуждение функций потерь в контексте нейронных сетей *см*. в работах LeCun and Huang [2005], LeCun et al. [2006] и Bengio et al. [2016].

4.6. Регуляризация и прореживание

Многослойные сети могут быть большими и иметь много параметров, что делает их чрезвычайно уязвимыми для переобучения. Регуляризация модели так же, если не больше, важна в глубоких нейронных сетях, как в линейных моделях. Регуляризаторы, рассмотренные в разделе 2.7.2, L_2, L_1 и эластичная сеть в полной мере относятся и к нейронным сетям. В частности, L_2-регуляризация, называемая также *снижением весов*, позволяет эффективно достигать хорошей обобщаемости во многих случаях, при этом рекомендуется настраивать величину регуляризации λ.

Еще одна эффективная техника для предотвращения переобучения нейронной сети называется *обучением с прореживанием* (dropout training) [Hinton et al., 2012, Srivastava et al., 2014]. Метод прореживания придуман для того, чтобы обучение сети не полагалось на конкретные веса. Идея в том, чтобы случайным образом удалить (обнулить) половину нейронов в сети (или в отдельном ее слое) для каждого обучающего примера при обучении методом стохастического градиентного спуска. Например, рассмотрим многослойный перцептрон с двумя скрытыми слоями (МСП2):

$$\begin{aligned}
\mathrm{NN}_{\mathrm{MLP2}}(\boldsymbol{x}) &= \boldsymbol{y};\\
\boldsymbol{h}^1 &= g^1(\boldsymbol{x}\boldsymbol{W}^1 + \boldsymbol{b}^1);\\
\boldsymbol{h}^2 &= g^2(\boldsymbol{h}^1\boldsymbol{W}^2 + \boldsymbol{b}^2);\\
\boldsymbol{y} &= \boldsymbol{h}^2\boldsymbol{W}^3.
\end{aligned}$$

Применяя обучение с прореживанием к МСП2, мы случайным образом обнуляем некоторые значения $\boldsymbol{h}^1$ и $\boldsymbol{h}^2$ на каждом раунде обучения:

$$\begin{aligned}
\mathrm{NN}_{\mathrm{MLP2}}(\boldsymbol{x}) &= \boldsymbol{y};\\
\boldsymbol{h}^1 &= g^1(\boldsymbol{x}\boldsymbol{W}^1 + \boldsymbol{b}^1);\\
\boldsymbol{m}^1 &\sim \mathrm{Bernouli}(r^1);\\
\tilde{\boldsymbol{h}}^1 &= \boldsymbol{m}^1 \odot \boldsymbol{h}^1;\\
\boldsymbol{h}^2 &= g^2(\boldsymbol{h}^1\boldsymbol{W}^2 + \boldsymbol{b}^2);\\
\boldsymbol{m}^2 &\sim \mathrm{Bernouli}(r^2);\\
\tilde{\boldsymbol{h}}^2 &= \boldsymbol{m}^2 \odot \boldsymbol{h}^2;\\
\boldsymbol{y} &= \tilde{\boldsymbol{h}}^2\boldsymbol{W}^3.
\end{aligned} \tag{4.7}$$

Здесь $\boldsymbol{m}^1$ и $\boldsymbol{m}^2$ – случайные *векторы-маски* такой же размерности, как $\boldsymbol{h}^1$ и $\boldsymbol{h}^2$ соответственно, а $\odot$ – операция поэлементного умножения. Значения элементов векторов-масок равны 0 или 1 и выбираются из распределения Бернулли с параметром r (обычно берут $r = 0{,}5$). Затем значения, соответствующие нулям, в векторах-масках обнуляются, так что скрытый слой $\boldsymbol{h}$ заменяется слоем $\tilde{\boldsymbol{h}}$ до передачи значений следующему слою.

В работе Wager et al. [2013] установлена тесная связь между методом прореживания и L_2-регуляризацией. В работе [Srivastava et al., 2014] описан другой взгляд, связывающий прореживание с методами усреднения моделей и ансамблевого обучения.

Метод прореживания – один из ключевых факторов, благодаря которым нейронные сети достигают высокого качества на задачах классификации изображений [Krizhevsky et al., 2012], особенно в сочетании с блоками активации ReLU [Dahl et al., 2013]. Он также эффективен в приложениях нейронных сетей к NLP.

4.7. Слои вычисления сходства и расстояния

Иногда требуется вычислить скалярное значение, отражающее *сходство*, *совместимость* или *расстояние* между двумя векторами. Например, векторы $\boldsymbol{v}_1 \in \mathbb{R}^d$ и $\boldsymbol{v}_2 \in \mathbb{R}^d$ могут быть выходными слоями двух МСП, и требуется обучить сеть так, чтобы она порождала похожие векторы для одних обучающих примеров и непохожие – для других.

Ниже описаны распространенные функции, которые принимают два вектора $\boldsymbol{u} \in \mathbb{R}^d$ и $\boldsymbol{v} \in \mathbb{R}^d$ и возвращают скаляр. Эти функции можно интегрировать с нейронной сетью прямого распространения (нередко так и поступают).

СКАЛЯРНОЕ ПРОИЗВЕДЕНИЕ Очень часто используют скалярное произведение:

$$\text{sim}_{\text{dot}}(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{u} \cdot \boldsymbol{v} = \sum_{i=1}^{d} \boldsymbol{u}_{[i]} \boldsymbol{v}_{[i]}. \tag{4.8}$$

ЕВКЛИДОВО РАССТОЯНИЕ Еще один популярный вариант – евклидово расстояние:

$$\text{dist}_{\text{euclidean}}(\boldsymbol{u}, \boldsymbol{v}) = \sqrt{\sum_{i=1}^{d} (\boldsymbol{u}_{[i]} - \boldsymbol{v}_{[i]})^2} = \sqrt{(\boldsymbol{u} - \boldsymbol{v}) \cdot (\boldsymbol{u} - \boldsymbol{v})} = \|\boldsymbol{u} - \boldsymbol{v}\|_2. \tag{4.9}$$

Заметим, что это метрика расстояния, а не сходства: малые (близкие к нулю) значения означают, что векторы похожи, а большие – что не похожи. Квадратный корень часто опускают.

ОБУЧАЕМЫЕ ФОРМЫ Скалярное произведение и евклидово расстояние – фиксированные функции. Но иногда требуется использовать параметрическую функцию, которую можно обучить порождению нужных значений сходства (или несходства) в зависимости от определенных измерений векторов. Популярной обучаемой функцией сходства является *билинейная форма*:

$$\text{sim}_{\text{bilinear}}(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{u}\boldsymbol{M}\boldsymbol{v}, \quad \boldsymbol{M} \in \mathbb{R}^{d \times d}, \tag{4.10}$$

где матрица $\boldsymbol{M}$ – параметр, нуждающийся в обучении.

Аналогично можно предложить обучаемую функцию расстояния:

$$\text{dist}(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u} - \boldsymbol{v})\boldsymbol{M}(\boldsymbol{u} - \boldsymbol{v}). \tag{4.10}$$

Наконец, многослойный перцептрон с одним выходным нейроном также можно использовать для порождения скаляра по двум векторам, подав ему на вход конкатенацию обоих векторов.

4.8. Слои погружения

Как будет сказано в главе 8, если на вход нейронной сети подаются категориальные признаки (например, признаки, принимающие одно из k различных символьных значений, скажем, слов из словаря), то часто с каждым возможным значением признака (в данном случае – словом) ассоциируют d-мерный вектор при некотором d. Эти векторы считаются параметрами модели и обучаются вместе с другими параметрами. Отображение символьного значения признака, например «слово номер 1249» на d-мерный вектор осуществляет *слой погружения* (иногда его еще называют *слой сопоставления*). Параметры слоя погружения организованы в виде матрицы $\boldsymbol{E} \in \mathbb{R}^{|\text{vocab}| \times d}$, строки которой соответствуют словам из словаря. Тогда операция сопоставления – это просто индексирование: $v_{1249} = \boldsymbol{E}_{[1249,:]}$. Если символьный признак кодируется унитарным вектором $\boldsymbol{x}$, то операцию сопоставления можно реализовать как умножение $\boldsymbol{x}\boldsymbol{E}$.

Векторы слов часто конкатенируются до передачи следующему слою. Погружения подробно рассматриваются в главе 8 при обсуждении плотных представлений категориальных признаков и в главе 10 при обсуждении предобученных представлений слов.

Глава 5

Обучение нейронной сети

Как и линейные модели, нейронная сеть состоит из дифференцируемых параметрических функций и обучается методами градиентной оптимизации (*см.* раздел 2.8). Целевая функция нелинейной нейронной сети невыпуклая, поэтому градиентный метод может застрять в локальном минимуме. Тем не менее на практике градиентные методы дают хорошие результаты.

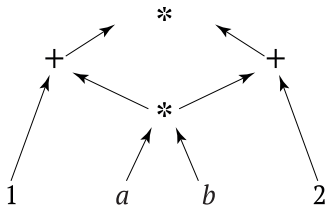
В основе этого подхода лежит вычисление градиента. Математика вычисления градиента для нейронной сети такая же, как для линейных моделей – это не что иное, как правило дифференцирования сложной функции. Но для сложных сетей процесс может оказаться трудоемким и подверженным ошибкам. По счастью, градиенты можно эффективно и автоматически вычислять с помощью *алгоритма обратного распространения* [LeCun et al., 1998b, Rumelhart et al., 1986]. Это просто заумное название для методического вычисления производных сложного выражения по правилу дифференцирования сложной функции с кешированием промежуточных результатов. В общем случае алгоритм обратного распространения представляет собой частный случай алгоритма автоматического дифференцирования с обратным накоплением [Neidinger, 2010, раздел 7], [Baydin et al., 2015, Bengio, 2012]. В следующем режиме описано автоматическое дифференцирование в обратном режиме, а оставшаяся часть главы посвящена практическим приемам обучения нейронных сетей.

5.1. Абстракция графа вычислений

Хотя градиенты по различным параметрам сети можно вычислять вручную и реализовать это в коде, такая процедура громоздка и подвержена ошибкам. Обычно предпочтительнее использовать автоматические инструменты вычисления градиента [Bengio, 2012]. Абстракция графа вычислений позволяет легко строить произвольные сети, вычислять их предсказания для заданных входных данных на прямом проходе и вычислять градиенты произвольной скалярной потери по параметрам на обратном проходе.

Граф вычислений – это представление произвольного математического вычисления в виде графа. Это ориентированный ациклический граф (ОАГ), вершины которого соответствуют математическим операциям или (связанным) переменным, а ребра – потоку промежуточных значений между вершинами. Структура графа определяет порядок вычислений в терминах зависимостей между компонентами. Граф представляет собой ОАГ, а не дерево, потому что результат одной

операции может подаваться на вход нескольких следующих за ней. Рассмотрим, к примеру, граф вычисления $(a * b + 1) * (a * b + 2)$:



Вычисление $a * b$ общее. Мы ограничиваемся случаем, когда граф вычислений связный (в несвязном графе каждая компонента связности – независимая функция, которую можно вычислять и дифференцировать независимо от других компонент связности).

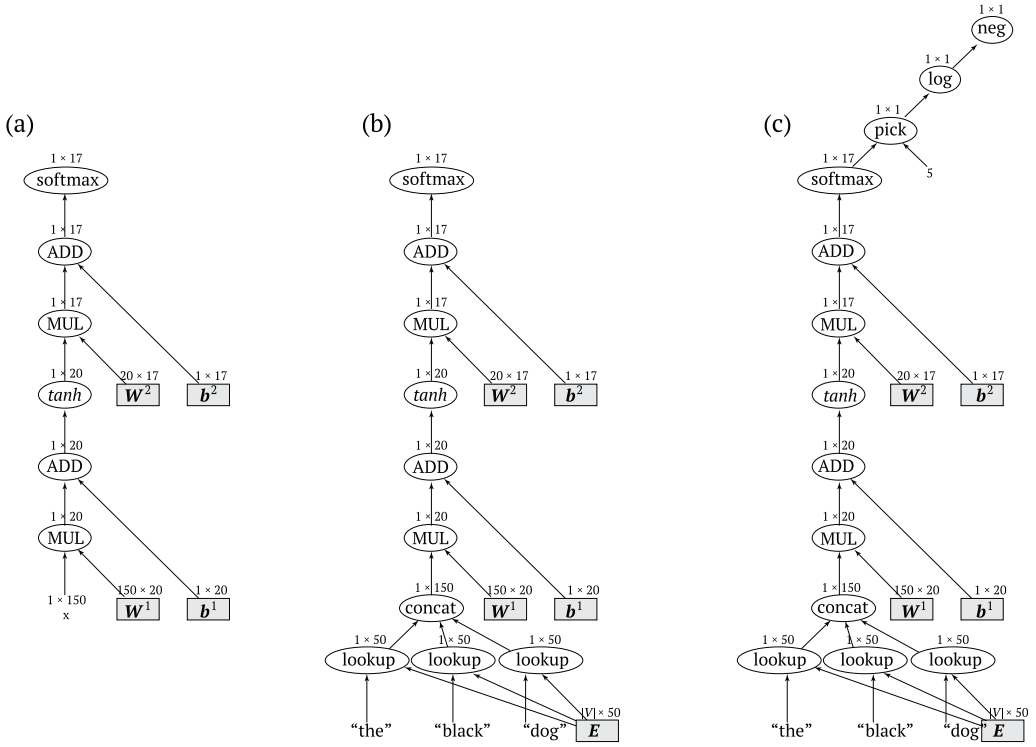


Рис. 5.1 ❖ (a) Граф с несвязанным входом; (b) граф с конкретным входом; (c) граф с конкретным входом, ожидаемым выходом и конечной вершиной потери

Поскольку нейронная сеть по сути своей – математическое выражение, ее можно представить в виде графа вычислений. Например, на рис. 5.1a показан граф вычислений для МСП с одним скрытым слоем и выходным преобразованием softmax. В нашей нотации овальные узлы представляют математические операции или функции, а закрашенные серым прямоугольные узлы – параметры (связанные переменные). Входы сети считаются константами и изображаются без

какой-либо рамки. Для входных и параметрических вершин нет входящих ребер, а для выходных вершин нет исходящих ребер. Выход каждого узла – матрица, размер которой указан над узлом.

Этот граф неполон: не задав входов, мы не сможем вычислить выходы. На рис. 5.1b показан полный граф для МСП, который принимает три слова на входе и предсказывает распределение меток частей речи для третьего слова. Этот граф можно использовать для предсказания, но не для обучения, поскольку на выходе выдается вектор (а не скаляр), и граф не учитывает ни правильного ответа, ни потери. Наконец, на рис. 5.1c показан граф вычислений для одного обучающего примера, входами которого являются погружения слов «the», «black», «dog», а ожидаемым выходом – метка «NOUN» (с индексом 5). Узел *pick* реализует операцию индексирования: он получает вектор и индекс (в данном случае – 5) и возвращает соответствующий элемент вектора.

После того как граф построен, уже не составляет труда выполнить либо прямое вычисление (получить результат вычисления), либо обратное вычисление (вычислить градиенты), что мы и продемонстрируем ниже. Построение графов может поначалу напугать, но в действительности это делается очень просто с помощью специальных программных библиотек и API.

5.1.1. Прямое вычисление

На прямом проходе вычисляются выходы узлов графа. Поскольку выход узла зависит только от него самого и входящих в него ребер, для вычисления выходов всех узлов нужно просто обойти узлы в топологическом порядке и вычислить выход каждого, зная уже вычисленные выходы его предшественников.

Формально, если имеется граф с N узлами, то мы ассоциируем с каждым узлом индекс i, равный его номеру в топологическом порядке обхода. Пусть f_i – функция, вычисленная узлом i (например, *умножение*, *сложение* и т. д.). Обозначим $\pi(i)$ – множество родителей узла i, а $\pi^{-1}(i) = \{j \mid i \in \pi(j)\}$ – множество дочерних узлов узла i (они являются аргументами f_i). Обозначим $v(i)$ – выход узла i, т. е. результат применения f_i к выходным значениям ее аргументов $\pi^{-1}(i)$. Для узлов переменных и входных узлов f_i – постоянная функция и множество $\pi^{-1}(i)$ пусто. На прямом проходе по графу вычислений вычисляется $v(i)$ для всех $i \in [1, N]$.

Алгоритм 5.3. Прямой проход по графу вычислений

1: **for** $i = 1$ to N **do**
2: Положить $a_1, \ldots, a_m = \pi^{-1}(i)$
3: $v(i) \leftarrow f_i(v(a_1), \ldots, v(a_m))$

5.1.2. Вычисление на обратном проходе (производные, обратное распространение)

Обратный проход начинается с назначения узла N со скалярным выходом (1 × 1) узлом потери и прогона прямого вычисления до этого узла. В процессе обратного вычисления вычисляются градиенты значения этого узла по параметрам. Обозначим $d(i)$ величину $\frac{\partial N}{\partial N}$. Алгоритм обратного распространения применяется для

вычисления значений $d(i)$ для всех узлов i. На обратном проходе заполняется таблица значений $d(1), \ldots, d(N)$, как показано в алгоритме 5.4.

Алгоритм 5.4. Обратный проход по графу вычислений (обратное распространение)

1: $d(N) \leftarrow 1$ ▷ $\frac{\partial N}{\partial N} = 1$

2: **for** $i = \mathrm{N} - 1$ to 1 **do**

3: $\quad d(i) \leftarrow \sum_{j \in \pi(i)} d(j) \cdot \frac{\partial f_j}{\partial i}$ $\quad \frac{\partial N}{\partial i} = \sum_{j \in \pi(i)} \frac{\partial N}{\partial j} \frac{\partial j}{\partial i}$

Алгоритм обратного распространения (алгоритм 5.4) – это, по существу, правило дифференцирования сложной функции. Величина $\frac{\partial f_j}{\partial i}$ – частная производная $f_j(\pi^{-1}(j))$ по аргументу $i \in \pi^{-1}(j)$. Это значение зависит от функции f_j и значений $v(a_1), \ldots, v(a_m)$ (где $a_1, \ldots, a_m = \pi^{-1}(j)$) ее аргументов, которые были вычислены на прямом проходе.

Таким образом, для определения нового вида узла необходимо определить два метода: один для вычисления прямого значения $v(i)$, зная входы в узел, а второй для вычисления $\frac{\partial f_i}{\partial x}$ для всех $x \in \pi^{-1}(i)$.

ПРОИЗВОДНЫЕ «НЕМАТЕМАТИЧЕСКИХ» ФУНКЦИЙ Если определение $\frac{\partial f_i}{\partial x}$ для математических функций типа log или + очевидно, то интерпретация производной от таких операций, как *pick*($\boldsymbol{x}$, 5), у некоторых вызывает затруднения. Нужно просто рассуждать в терминах вклада в вычисление. После того как i-й элемент вектора выбран, только он и принимает участие в последующем вычислении. Следовательно, градиент *pick*($\boldsymbol{x}$, 5) – это вектор $\boldsymbol{g}$ такой же размерности, как $\boldsymbol{x}$, в котором $\boldsymbol{g}_{[5]} = 1$ и $\boldsymbol{g}_{[i \neq 5]} = 0$. Аналогично для функции max(0, x) значение градиента равно 1 для $x > 0$ и 0 в противном случае.

Дополнительные сведения об автоматическом дифференцировании *см*. в работе Neidinger [2010, раздел 7] и Baydin et al. [2015]. Углубленное обсуждение алгоритма обратного распространения *см*. в книге Bengio et al. [2016, раздел 6.5] и работах Bengio [2012], LeCun et al. [1998b]. Популярная, но все-таки технически строгая презентация имеется в статье Криса Ола (Chris Olah) по адресу **http://colah.github.io/posts/2015-08-Backprop/**.

5.1.3. Программное обеспечение

Существует несколько программных пакетов, реализующих модель графа вычислений, в том числе: Theano[1] [Bergstra et al., 2010], TensorFlow[2] [Abadi et al., 2015]

[1] http://deeplearning.net/software/theano/.

[2] https://www.tensorflow.org/.

Chainer[1] и DyNet[2] [Neubig et al., 2017]. Все они поддерживают основные компоненты (типы узлов), необходимые для определения широкого круга нейросетевых архитектур, включая и структуры, описанные в этой книге. Создание графов почти прозрачно благодаря использованию перегрузки операторов. Каркас определяет типы для представления узлов графа (обычно называемых *выражениями*), методы конструирования узлов для выходных данных и параметров, а также набор функций и математических операций, которые принимают выражения и возвращают более сложные выражения. Ниже приведен пример кода на Python для создания графа вычислений, показанного на рис. 5.1c, с помощью пакета DyNet:

```
import dynet as dy
# Инициализация модели.
model = dy.Model ( )
mW1 = model.add_parameters((20, 150))
mb1 = model.add_parameters(20)
mW2 = model.add_parameters((17, 20))
mb2 = model.add_parameters(17)
lookup = model.add_lookup_parameters((100, 50))
trainer = dy.SimpleSGDTrainer(model)

def get_index (x) :
    pass # код опущен.
# Отображает слова на числовые идентификаторы.

# Следующий далее код строит и выполняет граф вычислений и обновляет параметры модели.
# Показана только одна точка, на практике этот код следует выполнять в цикле чтения данных.
# Построение графа вычислений:
dy.renew_cg( ) # создать граф.

# Обернуть параметры модели узлами графа
W1 = dy.parameter(mW1)
b1 = dy.parameter(mb1)
W2 = dy.parameter(mW2)
b2 = dy.parameter(mb2)

# Сгенерировать слой погружений.
vthe   = dy.lookup[get_index("the")]
vblack = dy.lookup[get_index("black")]
vdog   = dy.lookup[get_index("dog")]

# Соединить листовые узлы в полный граф.
x = dy.concatenate([vthe, vblack, vdog])
output = dy.softmax(W2*(dy.tanh(W1*x + b1)) + b2 )
loss = -dy.log(dy.pick(output, 5))
loss_value = loss.forward( )
loss.backward( )  # Градиент вычислен и сохранен в соответствующих параметрах.
trainer.update( ) # Обновить параметры в соответствии с градиентами.
```

Большая часть кода занимается разного рода инициализацией: в первом блоке определены параметры модели, общие для различных графов вычислений (напомним, что каждый граф соответствует одному обучающему примеру). Во вто-

[1] http://chainer.org.

[2] https://github.com/clab/dynet.

ром блоке параметры модели преобразуются в узлы графа (выражения). В третьем блоке извлекаются выражения для погружений входных слов. Наконец, в четвертом блоке создается граф. Обратите внимание, насколько прозрачно создание графа – существует почти взаимно однозначное соответствие между созданием графа и его математическим описанием. В последнем блоке показаны прямой и обратный проходы. А вот как выглядит эквивалентный код для пакета TensorFlow[1].

```
import tensorflow as
W1 = tf.get_variable("W1", [20, 150])
b1 = tf.get_variable("b1", [20])
W2 = tf.get_variable("W2", [17, 20])
b2 = tf.get_variable("b2", [17])
lookup = tf.get_variable("W", [100, 50])

def get_index (x) :
    pass # Logic omitted

p1 = tf.placeholder(tf.int32, [])
p2 = tf.placeholder(tf.int32, [])
p3 = tf.placeholder(tf.int32, [])
target = tf.placeholder (tf.int32, [])

v_w1 = tf.nn.embedding_lookup(lookup, p1)
v_w2 = tf.nn.embedding_lookup(lookup, p2)
v_w3 = tf.nn.embedding_lookup(lookup, p3)

x = tf.concat([v_w1, v_w2, v_w3], 0)
output = tf.nn. softmax(
    tf.einsum("ij, j->i", W2, tf.tanh(
        tf.einsum ("ij, j->i", W1, x) + b1)) + b2)

loss = -tf.log( output[target])
trainer = tf.train.GradientDescentOptimizer(0.1).minimize(loss)

# Граф определен. Откомпилировать его и подать конкретные данные.
# Показана только одна точка, на практике этот код следует выполнять в цикле чтения данных.
with tf.Session() as sess :
    sess.run(tf.global_variables_initializer())
    feed_dict = {
        p1 : get_index("the") ,
        p2 : get_index("black"),
        p3 : get_index("dog"),
        target : 5
    }
    loss_value = sess.run(loss, feed_dict)
    # Обновить, вызов backward не нужен.
    sess.run(trainer, feed_dict)
```

Главное различие между DyNet (и Chainer) и TensorFlow (и Theano) заключается в том, что в первых двух пакетах применяется *динамическое построение графа*, а в двух последних – *статическое построение графа*. При динамическом построении граф вычисления создается заново для каждого обучающего примера, для этого пишется код на объемлющем языке. После этого к графу применяются пря-

[1] Код для TensorFlow предоставил Тим Роктэшел (Tim Rocktäschel). Спасибо, Тим!

мое и обратное распространения. При статическом же построении форма графа вычислений определяется один раз в начале вычисления, для чего существует специальный API, а места входных и выходных значений занимают маркеры. Затем компилятор графов порождает оптимизированный граф вычислений, которому один за другим подаются обучающие примеры. Шаг компиляции графа в статических пакетах (TensorFlow и Theano) – одновременно благословение и проклятие. С одной стороны, будучи один раз откомпилирован, большой граф может эффективно выполняться на CPU или на GPU, что идеально в случае графов с фиксированной структурой, когда изменяются только входные данные. Но сам шаг компиляции может оказаться дорогим, а интерфейс для работы с ним более громоздкий. С другой стороны, в динамических пакетах во главу угла ставится построение больших динамических графов вычислений и выполнение их «на лету», без шага компиляции. Хотя теоретически скорость выполнения может оказаться ниже, чем в статических пакетах, на практике динамические пакеты вполне конкурентоспособны. Динамические пакеты особенно удобны для работы с рекуррентными и рекурсивными сетями, описанными в главах 14 и 18, а также для структурного предсказания, описанного в главе 19, когда графы для различных примеров имеют разные формы. В работе Neubig et al. [2017] можно найти дальнейшее сравнение динамического и статического подходов и результаты тестов производительности для различных пакетов. Наконец, пакеты типа Keras[1] предлагают высокоуровневый интерфейс, надстроенный над такими пакетами, как Theano и TensorFlow, что позволяет уложить определение и обучение сложных нейронных сетей в еще меньшее число строк кода при условии, что архитектура настолько «устаканилась», что ее поддержка включена в высокоуровневый интерфейс.

5.1.4. Рецепт реализации

В алгоритме 5.5 показан псевдокод алгоритма обучения сети, в котором используется абстракция графа вычислений и динамическое построение графа.

Здесь `build_computation_graph` – определенная пользователем функция, которая строит граф вычислений для заданных входа, выхода и структуры сети и возвращает единственный узел потери. Функция `update_parameters` – правило обновления, зависящее от оптимизатора. Согласно рецепту граф создается для каждого обучающего примера. Это охватывает и случаи, когда структура сети меняется от примера к примеру, как в рекуррентных и в рекурсивных сетях (*см.* главы 14–18). Для сетей с фиксированной структурой, например МСП, эффективнее было бы создать один базовый граф вычислений и для каждого примера менять только входы и ожидаемые выходы.

Алгоритм 5.5. Обучение нейронной сети с помощью абстракции графа вычислений (с мини-пакетами размера 1)

1: Определить параметры сети
2: **for** iteration = 1 to T **do**
3: **for** обучающий пример $\boldsymbol{x}_i$, $\boldsymbol{y}_i$ из набора данных **do**
4: loss_node ← build_computation_graph($\boldsymbol{x}_i$, $\boldsymbol{y}_i$, parameters)

[1] https://keras.io.

```
        loss_node.forward()
        gradients ← loss_node().backward()
        parameters ← update_parameters(parameters, gradients)
return parameters
```

5.1.5. Композиция сети

При условии что выходом сети является вектор (матрица размера $1 \times k$), образовать композицию сетей тривиально – нужно лишь подать выход одной сети на вход другой, и тогда мы сможем строить произвольные сети. В абстракции графа вычислений эта возможность реализована явно: узел графа вычислений сам может являться графом вычислений с заданным выходным узлом. Таким образом можно проектировать сети произвольной глубины и сложности, а благодаря автоматическому прямому проходу и вычислению градиента на обратном проходе вычислять и обучать их нетрудно. Это позволяет определять и обучать изощренные рекуррентные и рекурсивные сети, рассматриваемые в главах 14–16 и 18, а также сети со структурным выходом и предназначенные для многоцелевого обучения (главы 19 и 20).

5.2. Практические вопросы

После того как проблема вычисления градиента решена, сеть можно обучить с помощью СГС или другого градиентного алгоритма оптимизации. Подлежащая оптимизации функция невыпукла, и в течение долгого времени обучение нейронной сети считалось «черной магией», доступной лишь немногим избранным. Действительно, на процесс оптимизации влияет много параметров, которые необходимо тщательно настраивать. И хотя эта книга не задумывалась как полное руководство по успешному обучению нейронных сетей, мы все же перечислим несколько наиболее важных моментов. Подробнее о методах и алгоритмах оптимизации для нейронных сетей *см*. в Bengio et al. [2016, глава 8]. Теоретическое обсуждение и анализ можно найти в работе Glorot and Bengio [2010]. Различные практические советы и рекомендации *см*. в работах Bottou [2012], LeCun et al. [1998a].

5.2.1. Выбор алгоритма оптимизации

Алгоритм СГС работает хорошо, но иногда медленно сходится. В разделе 2.8.3 перечислены некоторые альтернативные, более передовые алгоритмы на основе стохастического градиентного спуска. Большинство программных пакетов для работы с нейронными сетям предлагает реализации этих алгоритмов, поэтому экспериментировать с разными вариантами легко и зачастую имеет прямой смысл. Мы с коллегами обнаружили, что для обучения больших сетей алгоритм Adam [Kingma and Ba, 2014] очень эффективен и довольно устойчив к выбору скорости обучения.

5.2.2. Инициализация

Невыпуклость целевой функции означает, что процедура оптимизации может застрять в локальном минимуме или седловой точке и что, отправляясь из разных начальных точек (например, задавая разные значения параметров), можно прий-

ти к разным результатам. Поэтому рекомендуется выполнить обучение несколько раз с разными случайно выбранными начальными значениями и выбрать наилучший результат, сверяясь с контрольным набором[1]. Насколько будут различаться результаты вследствие разного выбора начального значения генератора случайных чисел, зависит от сети и набора данных, заранее это предсказать невозможно.

Выбор случайных значений оказывает сильное влияние на успех обучения. В эффективной схеме, описанной в работе Glorot and Bengio [2010] и названной инициализацией Ксавье по имени Глорота, предлагается следующая матрица весов $\boldsymbol{W} \in \mathbb{R}^{d_{in} \times d_{out}}$:

$$\boldsymbol{W} \sim \boldsymbol{U}\left[-\frac{\sqrt{6}}{\sqrt{d_{in}+d_{out}}}, +\frac{\sqrt{6}}{\sqrt{d_{in}+d_{out}}}\right], \tag{5.1}$$

где $U[a, b]$ – равномерное распределение в диапазоне $[a, b]$. Эта рекомендация основана на свойствах функции активации tanh, хорошо работает во многих ситуациях и многими принимается как метод инициализации по умолчанию.

Анализ, проведенный в работе He et al. [2015], говорит, что при использовании нелинейностей вида ReLU веса следует инициализировать, производя выборку из нормального распределения с нулевым средним и стандартным отклонением $\sqrt{\frac{2}{d_{in}}}$. Авторы показали, что эта инициализация работает лучше инициализации Ксавье в задаче классификации изображений, особенно когда используются глубокие сети.

5.2.3. Перезапуск и ансамбли

При обучении сложных сетей конечное решение часто зависит от случайной инициализации, и верность разных решений различается. Поэтому, если располагаемые вычислительные ресурсы позволяют, рекомендуется прогнать процесс обучения несколько раз с различными начальными весами и выбрать то решение, которое показывает наилучший результат на контрольном наборе. Эта техника называется *случайным перезапуском*. Верность модели, усредненная по всем выборам начального значения генератора, также представляет интерес, поскольку дает представление об устойчивости процесса.

Конечно, необходимость «настраивать» начальное значение генератора случайных чисел для инициализации модели подчас вызывает раздражение, но, с другой стороны, это дает простой способ получения разных моделей для решения одной и той же задачи, что открывает возможность использовать *ансамбли моделей*. Если имеется несколько моделей, то в основу предсказания можно положить результаты ансамбля моделей, а не одной-единственной модели (например, проведя голосование среди моделей или усреднив выходные векторы и взяв среднее в качестве выходного вектора ансамблевой модели). Использование ансамблей зачастую повышает верность предсказания ценой выполнения процедуры предсказания несколько раз (по разу для каждой модели).

---

[1] Во время отладки и если требуется воспроизводимость результатов, рекомендуется задавать одно и то же начальное значение генератора случайных чисел.

5.2.4. Исчезающие и взрывающиеся градиенты

В глубоких сетях часто бывает, что в процессе обратного распространения по графу вычислений градиент ошибки либо исчезает (приближается вплотную к нулю), либо «взрывается» (становится несообразно большим). Проблема осложняется в более глубоких сетях, особенно в рекурсивных и в рекуррентных [Pascanu et al., 2012]. Вопрос о том, как бороться с проблемой исчезающего градиента, все еще открыт. В качестве решения предлагается уменьшать глубину сети, производить пошаговое обучение (сначала обучить нижние слои, основываясь на каком-то вспомогательном выходном сигнале, а затем зафиксировать их и обучить верхние слои полной сети на основе реального сигнала, характерного для задачи), выполнять пакетную нормировку [Ioffe and Szegedy, 2015] (для каждого пакета нормировать входы в каждый слой сети на нулевое среднее и единичную дисперсию) или использовать специализированные архитектуры, спроектированные специально, чтобы улучшить поток градиента (например, архитектуры LSTM и GRU для рекуррентных сетей (*см.* главу 15)). Для борьбы со взрывающимся градиентом есть простое, но очень эффективное решение: отсекать градиенты, если их норма превышает заданный порог. Обозначим $\hat{\boldsymbol{g}}$ градиенты по всем параметрам сети, а $||\hat{\boldsymbol{g}}||$ – их норму L_2. В работе Pascanu et al. [2012] предлагается положить $\hat{\boldsymbol{g}} \leftarrow \frac{\text{threshold}}{\|\hat{\boldsymbol{g}}\|}\hat{\boldsymbol{g}}$, если $||\hat{\boldsymbol{g}}|| > \text{threshold}$.

5.2.5. Насыщение и мертвые нейроны

Слои, в которых функцией активации является tanh или сигмоида, склонны к насыщению – все выходные значения оказываются близки к единице, верхней грани функции активации. У насыщенных нейронов градиенты очень малы, чего следует избегать. Слои с функцией активации ReLU не подвержены насыщению, но могут «умирать» – большая часть или даже все значения становятся отрицательными и обрезаются по нулю для всех входов, в результате чего градиент для этого слоя оказывается равен нулю. Если ваша сеть плохо обучается, рекомендуется последить, нет ли в ней слоев, в которых много насыщенных или мертвых нейронов. Причиной появления насыщенных нейронов является подача слишком больших значений на вход слоя. Этим можно управлять путем изменения начальных значений параметров, масштабирования диапазона входных значений или изменения скорости обучения. Мертвые нейроны появляются, когда все входящие в слой сигналы отрицательны (например, это может случиться после сильного обновления градиента). В таких ситуациях помогает уменьшение скорости обучения. В случае насыщенных слоев есть еще одна рекомендация: нормировать значения в насыщенном слое после активации, т. е. вместо $g(\boldsymbol{h}) = \tanh(\boldsymbol{h})$ взять $g(\boldsymbol{h}) = \tanh(\boldsymbol{h})/||\tanh(\boldsymbol{h})||$. Нормировка слоя – эффективное средство борьбы с насыщением, но дорого обходится в смысле вычисления градиента. Родственный прием – *пакетная нормировка* (*см.* Ioffe and Szegedy [2015]), когда значения функции активации в каждом слое нормируются так, чтобы в каждом мини-пакете среднее было равно 0, а дисперсия – 1. Метод пакетной нормировки стал основным компонентом эффективного обучения глубоких сетей в компьютерном зрении. На момент написания этой книги он был не так популярен в приложениях к обработке естественного языка.

5.2.6. Тасование

Порядок, в котором обучающие примеры подаются сети, важен. В описанной выше формулировке СГС на каждом шаге выбирается случайный пример. На практике в большинстве реализаций обучающий набор перебирается в случайном порядке, т. е. осуществляется случайный выбор без возвращения. Рекомендуется тасовать обучающие примеры перед каждым проходом по данным.

5.2.7. Скорость обучения

Выбор скорости обучения также важен. Если скорость обучения слишком велика, то сеть не сойдется к эффективному решению. Если же она слишком мала, то для сходимости потребуется очень много времени. Рекомендуется экспериментировать с диапазоном начальных скоростей обучения [0, 1], например: 0.001, 0.01, 0.1, 1. Следите за тем, как изменяется потеря со временем, и уменьшайте скорость обучения, как только потеря перестает улучшаться на зарезервированном контрольном наборе. Метод *планирования скорости обучения* заключается в уменьшении скорости как функции от числа обработанных мини-пакетов. Популярный способ планирования – делить начальную скорость обучения на номер итерации. В работе Léon Bottou [2012] рекомендуется использовать скорость обучения вида $\eta_t = \eta_0(1 + \eta_0\lambda t)^{-1}$, где η_0 – начальная скорость обучения, η_t – скорость обучения на t-м обучающем примере, а λ – дополнительный гиперпараметр. Рекомендуется также определять хорошее значение η_0 по небольшой выборке, прежде чем запускать алгоритм на всем наборе данных.

5.2.8. Мини-пакеты

Обновление параметров происходит либо после каждого обучающего примера (мини-пакеты размера 1), либо после обработки k обучающих примеров. Для некоторых проблем выгодно вести обучение на мини-пакетах большего размера. Если говорить в терминах абстракции графа вычислений, то можно создать граф вычислений для каждого из k обучающих примеров, а затем поместить все k узлов потерь под узел усреднения, выход которого и будет считаться потерей на мини-пакете. Обучение с большими мини-пакетами может оказаться выгодно и с точки зрения эффективности вычислений на компьютерах со специализированной архитектурой, например GPU, или за счет замены операций умножения вектора на матрицу операцией умножения матрицы на матрицу. Но эти вопросы выходят за рамки настоящей книги.

Часть II

РАБОТА С ДАННЫМИ ЕСТЕСТВЕННОГО ЯЗЫКА

Глава 6

Признаки для текстовых данных

В предыдущих главах мы обсуждали общую постановку задачи обучения и рассмотрели несколько моделей машинного обучения и алгоритмы для их обучения. Все эти модели принимают входные векторы $\boldsymbol{x}$ и возвращают предсказания. До сих пор мы предполагали, что векторы $\boldsymbol{x}$ заданы. В лингвистической обработке векторы $\boldsymbol{x}$ формируются на основе текстовых данных, чтобы отразить различные лингвистические свойства текста. Отображение текстовых данных на вещественные векторы называется *выделением признаков*, или *представлением в пространстве признаков*. Этим занимается *функция выделения признаков*. Решение о том, какие признаки выбрать, – неотъемлемая часть успешного проекта машинного обучения. Хотя глубокие нейронные сети в значительной степени освобождают человека от необходимости заниматься конструированием признаков, пристойный набор базовых признаков все же придется определить. Особенно это справедливо в отношении лингвистических данных, которые поступают в виде последовательности дискретных символов. Эту последовательность необходимо как-то преобразовать в числовой вектор, а как именно это сделать, не очевидно.

Теперь отвлечемся от механизмов обучения и обсудим, какие функции выделения признаков применяются при работе с лингвистическими данными. Этим мы будем заниматься на протяжении нескольких последующих глав.

В этой главе предлагается обзор типичных видов источников информации, доступных для использования в качестве признаков при работе с текстовыми лингвистическими данными. В главе 7 обсуждаются варианты признаков для некоторых конкретных проблем NLP. Глава 8 посвящена кодированию признаков в виде входных векторов, которые можно подать на вход нейронной сети.

6.1. Типология проблем классификации NLP

Вообще говоря, проблемы классификации в естественном языке можно отнести к нескольким крупным категориям в зависимости от того, какой объект классифицируется. (Некоторые проблемы NLP не укладываются ни в какую категорию, например проблемы, в которых требуется подождать предложения, или более длинные тексты, т. е. возникающие в процессе реферирования документов и в машинном переводе. Их мы обсудим в главе 17.)

Слово. В таких проблемах мы имеем слово, например «dog», «magnificent», «magnifficant» или «parlez», и должны что-то сказать о нем: обозначает ли оно живое существо? на каком языке оно написано? насколько оно распространено? какие слова похожи на него? является ли оно неправильно написанным другим словом? и т. д. Подобного рода проблемы встречаются крайне редко, поскольку слова почти никогда не встречаются изолированно, и для многих слов интерпретация зависит от контекста.

Тексты. В таких проблемах нам предъявлен фрагмент текста: фраза, предложение, абзац или документ, – и требуется что-то сказать о нем. Это спам или нет? Текст посвящен политике или спорту? Он саркастический? Он имеет положительную, отрицательную или нейтральную эмоциональную окраску (по отношению к некоторому предмету)? Кто его автор? Можно ли ему доверять? Какой фиксированный набор намерений отражает текст (возможно, никакой)? Понравится ли этот текст юношам от 16 до 18 лет? И т. д. Такого рода проблемы возникают очень часто, мы будем использовать для них собирательный термин: проблемы *классификации документов*.

Парные тексты. В таких проблемах дана пара слов или более длинных текстов и требуется что-то сказать о ней. Являются ли слова A и B синонимами? Является ли слово A правильным переводом слова B? Написаны ли документы A и B одним автором? Можно ли вывести смысл предложения A из предложения B?

Слово в контексте. Здесь дан фрагмент текста и некоторое слово (фраза, буква и т. п.) в нем, а требуется классифицировать слово в контексте текста. Например, является ли слово *book* в предложении *I want to book a flight* существительным, глаголом или прилагательным? Является ли слово *apple* в заданном контексте названием фрукта или компании? Правильно ли употребление предлога *on* в предложении *I read a book on London*? Означает ли данная точка конец предложения или является частью аббревиатуры? Является ли данное слово именем человека, названием места или организации? И т. д. Такого рода вопросы часто возникают как часть более крупных задач, например: аннотирование предложения метками частей речи, разбиение документа на предложения, поиск всех именованных сущностей в тексте, поиск всех документов, в которых упоминается заданная сущность, и т. д.

Связь между двумя словами. Даны два слова или фразы в контексте большего документа, и требуется что-то сказать о связях между ними. Является ли слово A подлежащим при глаголе в заданном предложении? Существует ли отношение «покупки» между словами A и B в заданном тексте? И т. д.

Многие задачи классификации такого типа можно обобщить на *структурные проблемы*, когда требуется принять несколько взаимосвязанных классификационных решений, причем одно решение может влиять на другие. Подобные проблемы обсуждаются в главе 19.

ЧТО ТАКОЕ СЛОВО? Мы употребляем термин *слово* вольно. На самом деле вопрос о том, что такое слово, является предметом споров в среде лингвистов, и ответ на него не всегда ясен.

Одно из определений (которому мы, хоть и не строго, следуем в этой книге) говорит, что слова – это последовательности букв, разделенные пробельными

символами. Это очень упрощенное определение. Во-первых, знаки препинания в английском языке не отделяются пробелами, поэтому, согласно нашему определению, `dog`, `dog?`, `dog.` и `dog`) – различные слова. Определение можно поправить, сказав, что слова разделяются пробельными символами или знаками препинания. Процесс *лексемизации* (tokenization) отвечает за разбиение текста на лексемы (то, что мы будем называть словами) на основе пробельных символов и знаков препинания. Лексемизатор для английского языка очень прост, хотя ему все же нужно принимать во внимание специальные случаи, например аббревиатуры (`I.B.M`) и обращения (`Mr.`), которые не нужно разбивать. В других языках все может быть гораздо сложнее: в иврите и арабском языке некоторые слова присоединяются к следующему без пробелов, а в китайском пробелов нет вовсе. И это лишь немногие примеры.

При работе с английским или похожим языком (как в этой книге) лексемизация по пробельным символам и знакам препинания (с учетом нескольких особых случаев) может дать неплохую аппроксимацию слов. Однако наше определение слова все еще носит технический характер: оно основано на том, как мы пишем. Другое (лучшее) определение заключается в том, что слово – «это мельчайшая единица смысла». Если следовать этому подходу, то мы увидим, что определение на основе пробелов проблематично. После разбиения по пробельным символам и знакам препинания все равно остаются последовательности, например `don't`, которые на самом деле содержат два слова, `do not`, объединенных в один символ. Английские лексемизаторы учитывают и такие случаи тоже. Символы `cat` и `Cat`[1] имеют одинаковый смысл, но являются ли они одним и тем же словом? А вот еще более интересный вопрос: такие последовательности, как `New York`, – это одно слово или два? А как насчет `ice cream`[2]? Это то же самое, что `ice-cream` или `icecream`? А что сказать про идиомы типа `kick the bucket`[3]?

Вообще говоря, мы различаем *слова* и *лексемы*. Выход лексемизатора мы будем называть *лексемами*, а единицы, несущие смысл, – *словами*. Лексема может состоять из нескольких слов, и иногда разные лексемы означают одно и то же слово.

При всем при том в этой книге термин *слово* употребляется очень вольно и считается синонимом *лексемы*. Однако важно иметь в виду, что на самом деле ситуация сложнее.

6.2. Признаки для проблем NLP

Ниже будут описаны наиболее распространенные признаки, используемые при решении поставленных выше проблем. Поскольку слова и буквы – это дискретные объекты, признаки будут иметь вид индикаторов или счетчиков. *Индикаторный признак* принимает значения 0 и 1 в зависимости от выполнения некоторого условия (например, признак, принимающий значение 1, если в документе хотя бы

[1] Кошка и имя Кэт. – *Прим. перев.*

[2] Различные написания слова «мороженое». – *Прим. перев.*

[3] Сыграть в ящик, умереть. – *Прим. перев.*

один раз встретилось слово *dog*, и 0 в противном случае). Значение *счетчика* зависит от того, сколько раз произошло некоторое событие, например можно рассмотреть признак, указывающий, сколько раз в тексте встречается слово *dog*.

6.2.1. Непосредственно наблюдаемые свойства

Признаки для одиночных слов. Если нас интересует слово вне контекста, то основным источником информации являются *буквы*, составляющие слово, и порядок их следования, а также выводимые из них свойства, например: длина слова, орфографические особенности слова (является ли первая буква заглавной? Все ли буквы заглавные? Есть ли в слове дефис? Встречаются ли в нем цифры? И т. д.), префиксы и суффиксы слова (начинается ли оно с *un*? Заканчивается ли на *ing*?). Можно также связать слово с внешними источниками информации: сколько раз это слово встречается в большом корпусе текстов? Встречается ли оно в списке распространенных в США личных имен? И т. д.

Леммы и основы. В этой книге мы часто рассматриваем *лемму* слова (словарную статью). Так, формам *booking*, *booked*, *books* соответствует общая лемма *book*. Обычно это отображение осуществляется с помощью лексических фондов лемм или морфологических анализаторов, доступных для многих языков. Лемма слова может быть неоднозначной, и лемматизация дает более точный результат, если известен контекст слова. Лемматизация – процесс, определенный лингвистически, поэтому она может плохо работать для форм, отсутствующих в фонде лемм или написанных неправильно. Существует также процесс *стемминга*, он более грубый, чем лемматизация, но работает для любой последовательности букв. Стеммер отображает последовательности букв в более короткие последовательности, основываясь на эвристических свойствах конкретного языка, например различные флексии одного слова отображаются на одну и ту же последовательность. Заметим, что результатом стемминга необязательно является существующее слово: слова `picture`, `pictures` и `pictured` отображаются в одну и ту же последовательность `pictur`. Существуют разные стеммеры, отличающиеся уровнем агрессивности преобразований.

Лексические ресурсы. Дополнительным источником информации о формах слов являются *лексические ресурсы*. По сути дела, это словари, предназначенные для доступа из программ, а не для чтения человеком. Лексический ресурс обычно содержит информацию о словах, их связях с другими словами, а также может предоставлять дополнительные сведения.

Например, для многих языков имеются лексические фонды – лексиконы, – которые сопоставляют флективным словоформам возможные результаты их морфологического анализа (например, говорят, что некоторое слово может быть как существительным женского рода множественного числа, так и глаголом в прошедшем времени). Такие лексиконы обычно включают и информацию о леммах.

Для английского языка хорошо известен лексический ресурс *WordNet* [Fellbaum, 1998]. Это очень большой составленный вручную набор данных, ставящий целью собрать в одном месте концептуальные семантические знания о словах. Каждое слово принадлежит одному или нескольким *синсетам*, каждый из которых описывает одно когнитивное понятие. Например, слово *star* – существительное, принадлежащее синсетам *astronomical celestial body* (астрономическое небесное тело), *someone who is dazzlingly skilled* (некто, обладающий выдающимися умениями), *any*

celestial body visible from earth (любое небесное светило, видимое с Земли), *an actor who plays a principle role* (актер, сыгравший главную роль) и другим. Второй из вышеупомянутых синсетов включает также слова *ace* (ас)*, adept* (искусник)*, champion* (чемпион)*, sensation* (бомба)*, maven* (знаток)*, virtuoso* (виртуоз) и другие. Синсеты связаны между собой семантическими отношениями, например гиперонимии и гипонимии (бóльшая и меньшая специфичность). Скажем, первый синсет слова *star* включает слова *sun* (солнце) и *nova* (новая) (гипонимы), а также *celestial body* (небесное тело) (гипероним). В WordNet есть и другие семантические отношения: антонимы (слова с противоположным смыслом), голонимы и меронимы (отношения часть–целое и целое–часть). WordNet содержит информацию о существительных, глаголах, прилагательных и наречиях.

FrameNet [Fillmore et al., 2004] и VerbNet [Kipper et al., 2000] – вручную составленные лексические ресурсы, сосредоточенные на глаголах. Для многих глаголов в них перечислены допускаемые ими аргументы (например, *giving* допускает базовые аргументы `Donor`, `Recipient` и `Theme` (нечто, что дается), и у многих глаголов могут быть небазовые аргументы, например `Time`, `Purpose`, `Place` и `Manner`.

База данных Paraphrase (PPDB) [Ganitkevitch et al., 2013, Pavlick et al., 2015] представляет собой большой, автоматически созданный набор парафраз. Он содержит список слов и фраз и для каждой из них – список слов и фраз, означающих приблизительно то же самое.

Подобные лексические ресурсы содержат огромный объем информации и могут служить прекрасным источником признаков. Однако способ эффективного использования этой символической информации зависит от задачи, и зачастую требуется незаурядная изобретательность или нетривиальные инженерные усилия. В настоящее время они редко используются в моделях нейронных сетей, но в будущем это может измениться.

Дистрибутивная информация. Еще один важный источник информации о словах – *дистрибутивность*: какие еще слова ведут себя похоже на данное в имеющемся тексте? Эта тема заслуживает особого обсуждения, которое мы отложим до раздела 6.2.5. В разделе 11.8 мы обсудим, как можно использовать лексические ресурсы для включения знаний в дистрибутивные векторы слов, порождаемые нейросетевыми алгоритмами.

ПРИЗНАКИ ДЛЯ ТЕКСТА При рассмотрении предложения, абзаца или документа наблюдаемыми признаками являются счетчики и порядок *букв* и *слов* в тексте.

Мешок слов. Для выделения признаков из предложений и документов очень часто применяется подход на основе мешка слов (bag-of-words – BOW). В этом случае мы строим гистограмму распределения слов в тексте, т. е. считаем счетчики всех слов признаками. Обобщение этого подхода со слов на «базовые элементы» – мешок буквенных биграмм – мы использовали в примере определения языка в разделе 2.3.1.

Можно также вычислять величины, непосредственно выводимые из слов и букв, например *длину* предложения в терминах количества букв или слов. При рассмотрении отдельных слов мы, конечно, можем использовать вышеупомянутые признаки, например подсчитывать количество слов с определенным префиксом или суффиксом в документе либо отношение количества коротких слов (длина меньше заданного порога) к количеству длинных слов в документе.

Веса. Как и раньше, мы можем включить статистику, основанную на внешней информации, например обратив внимание на слова, которые много раз встречаются в данном документе, но при этом относительно редко встречаются во внешнем наборе документов (тем самым мы отделим слова, которые часто встречаются в документах, потому что они вообще очень распространены, например *a* и *for*, от слов, которые часто встречаются, потому что относятся к теме документа). При использовании мешка слов популярна схема взвешивания TF-IDF [Manning et al., 2008, глава 6]. Рассмотрим документ d, являющийся частью корпуса D. Вместо того чтобы представлять каждое слово w в d нормированным счетчиком встречаемости в самом документе $\frac{\#_d(w)}{\sum_{w' \in d} \#_d(w')}$ (частотой терма – **T**erm **F**requency), в схеме TF-IDF оно представляется величиной $\frac{\#_d(w)}{\sum_{w' \in d} \#_d(w')} \times \log \frac{|D|}{|\{d \in D : w \in d\}|}$. Второй член называется обратной частотой документа (**I**nverse **D**ocument **F**requency) и равен числу, обратному количеству различных документов во всем корпусе, в которых встречается данное слово. Эта схема взвешивания позволяет выделить слова, характерные для данного текста.

Помимо отдельных слов, можно рассматривать пары или тройки соседних слов. Они называются *n-граммами*. Признаки на основе n-грамм обсуждаются в разделе 6.2.4.

ПРИЗНАКИ СЛОВ В КОНТЕКСТЕ При рассмотрении слова в предложении или в документе непосредственно наблюдаемыми признаками являются позиция слова в предложении, а также окружающие его слова или буквы. Слова, расположенные ближе к целевому слову, часто более информативны, чем слова, расположенные поодаль[1].

Окна. По этой причине нередко акцентируют внимание на ближайшем контексте слова, рассматривая окружающее его *окно* (т. е. по k слов с каждой стороны, где в качестве k обычно берут значение 2, 5 или 10), и в качестве признаков берут идентификаторы слов в окне (например, признаком будет «слово X встретилось в окне шириной 5 слов вокруг целевого слова»). Рассмотрим, к примеру, предложение *the brown fox jumped over the lazy dog* с целевым словом *jumped*. Окно шириной 2 слова порождает такой набор признаков: { слово=brown, слово=fox, слово=over, слово=the }. Подход на основе окон – разновидность мешка слов, только ограниченного объектами в маленьком окне.

Фиксированный размер окна дает возможность ослабить предположение о безразличности порядка в мешке слов и принять во внимание относительные позиции слов в окне. Это дает нам *позиционные признаки*, например «слово X встретилось двумя словами левее целевого слова». Так, в примере выше учет позиции в окне дает такой набор признаков: { слово-2=brown, слово-1=fox, слово+1=over, слово+2=the }.

Кодирование оконных признаков векторами обсуждается в разделе 8.2.1. В главах 14 и 16 мы познакомимся с архитектурой biRNN, которая обобщает оконные признаки, предлагая гибкое, настраиваемое и обучаемое окно.

---

[1] Отметим, что это чрезмерно общее утверждение, во многих языках имеют место «дальнодействующие» зависимости между словами: слово в конце текста вполне может зависеть от слова в его начале.

Позиция. Помимо контекста слова, нас может интересовать абсолютная позиция слова в предложении. Можно рассмотреть признаки вида «целевое слово является пятым в предложении» или его гистограммный вариант с более крупными категориями: слово входит в первый десяток, во второй десяток и т. д.

ПРИЗНАКИ, УЧИТЫВАЮЩИЕ ОТНОШЕНИЯ СЛОВ При рассмотрении двух слов в контексте мы можем в дополнение к позиции каждого слова и окружающих слов учесть расстояние между словами и *идентификаторы* слов, расположенных между ними.

6.2.2. Производные лингвистические свойства

Предложения естественного языка обладают структурой, не исчерпывающейся линейным порядком слов. Структура подчиняется сложному набору правил, который мы непосредственно не наблюдаем. Совокупность этих правил называется синтаксисом, а изучение их природы и закономерностей естественного языка – предмет лингвистики[1]. Хотя точная структура языка остается тайной, а правила, управляющие многими тонкими языковыми явлениями, либо еще не исследованы, либо являются предметом споров между лингвистами, некое подмножество механизмов, регулирующих использование языка, хорошо осмыслено и документировано. Сюда относятся такие концепции, как классы слов (метки частей речи), морфология, синтаксис и отчасти даже семантика.

Хотя лингвистические свойства текста невозможно наблюдать непосредственно в виде форм и порядка слов в предложениях, их можно вывести из предложения с той или иной точностью. Существуют специализированные системы для предсказания частей речи, синтаксических деревьев, семантических ролей, отношений в структуре дискурса и других лингвистических свойств с различной степенью точности[2], и эти предсказания зачастую служат хорошими признаками для решения проблем классификации более высокого уровня.

ЛИНГВИСТИЧЕСКОЕ АННОТИРОВАНИЕ Изучим некоторые формы лингвистических аннотаций. Рассмотрим предложение *the boy with the black shirt opened the door with a key*[3]. На нижнем уровне аннотирования каждому слову сопоставляется *часть речи*:

the	boy	with	the	black	shirt	opened	the	door	with	a	key
Det	Noun	Prep	Det	Adj	Noun	Verb	Det	Noun	Prep	Det	Noun

Поднимаясь вверх по цепочке, мы помечаем границы *синтаксических оборотов*, одним из которых является именная группа (NP) *the boy*:

[1] Последнее предложение, конечно, является сильным упрощением. Предмет лингвистики гораздо шире, чем синтаксис, существуют и другие системы, регулирующие лингвистическое поведение человека, помимо синтаксической. Но для целей этой вводной книги такого упрощенного понимания достаточно. Углубленный обзор можно найти в рекомендованной литературе в конце этой главы.

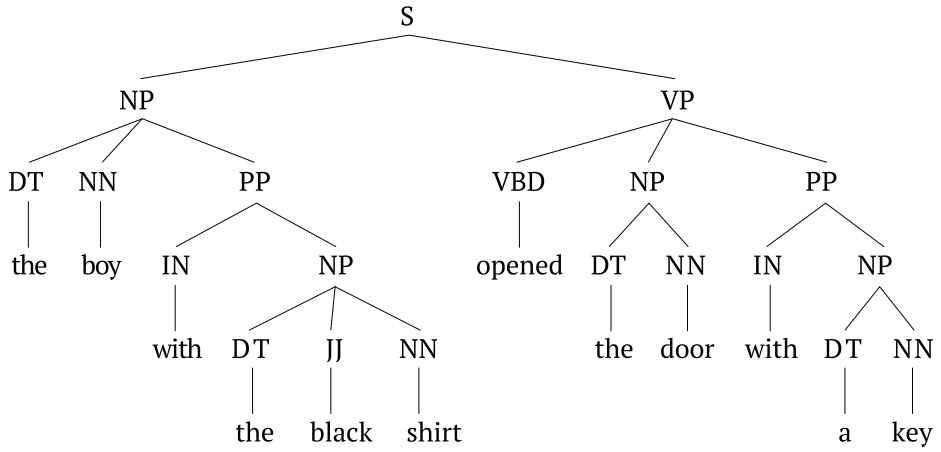
[2] На самом деле улучшение качества предсказания этих лингвистических свойств и есть проблема обработки естественного языка, которую пытаются решить многие исследователи.

[3] Мальчик в черной рубашке открыл дверь ключом. – *Прим. перев.*

[$_{NP}$ the boy] [$_{PP}$ with] [$_{NP}$ the black shirt] [$_{VP}$ opened] [$_{NP}$ the door] [$_{PP}$ with] [$_{NP}$ a key]

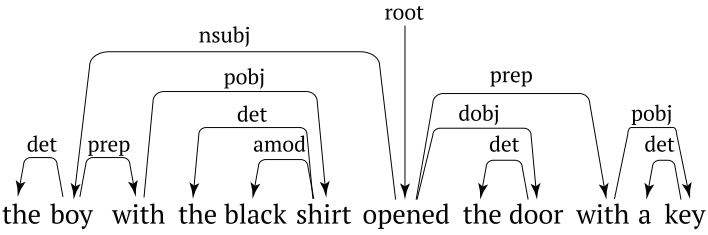
Заметим, что слово *opened* помечено как глагольный оборот (verbal-chunk – VP). Может показаться, что это не очень полезно, потому что мы и так знаем, что это глагол. Однако VP-обороты могут содержать больше элементов, например: *will open* или *did not open*.

Информация о синтаксических оборотах локальна. Более глобальная синтаксическая структура называется *деревом составляющих*, или *фразово-структурным деревом*.



Деревья составляющих представляют собой вложенную скобочную запись предложения, показывающую иерархию синтаксических блоков: именная группа *the boy with the black shirt* составлена из именной группы *the boy* и предложной группы (PP) *with the black shirt*. Последняя, в свою очередь, содержит именную группу *the black shirt*. Тот факт, что оборот *with a key* расположен под VP, а не под NP *the door*, говорит о том, что *with a key* модифицирует глагол *opened* (*opened with a key*), а не именную группу (*a door with a key*).

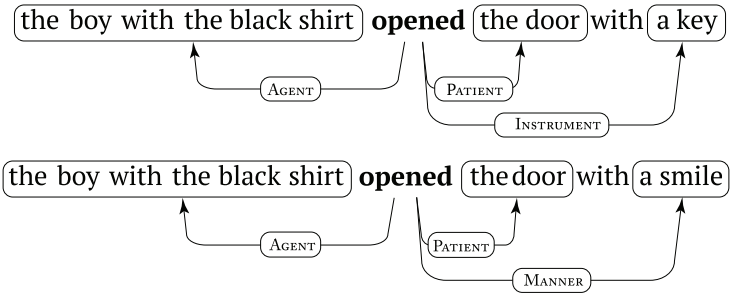
Другой вид синтаксической аннотации называется *деревом зависимостей*. В этом случае каждое слово предложения является модификатором другого слова, которое называется его заглавным словом. Заглавное слово есть у каждого слова предложения, кроме главного слова, обычно глагола, которое является корнем предложения и возглавляется специальным узлом root.



Если деревья составляющих проявляют группировку слов во фразы, то деревья зависимостей проявляют *отношения модификации* и *связи* между словами.

Слова, далеко отстоящие друг от друга в линейной форме предложения, могут оказаться близки в его дереве зависимостей. Например, в линейной форме слова *boy* и *opened* разделены четырьмя словами, но в дереве зависимостей их соединяет прямое ребро *nsubj*.

Отношения зависимости синтаксические: они касаются структуры предложения. Другие виды отношений в большей степени семантические. Рассмотрим, например, модификаторы глагола *open*, называемые также *аргументами* глагола. Синтаксическое дерево ясно помечает *the boy* (*with the black shirt)*, *the door* и *with a key* как аргументы и, кроме того, сообщает, что *with a key* является аргументом *open*, а не модификатором *door*. Однако оно ничего не говорит о *семантических ролях* аргументов по отношению к глаголу, т. е. что *the boy* – это агент (Agent), совершающий действие, а *a key* – инструмент (Instrument) (сравните с *the boy opened the door with a smile*[1]). Здесь предложение имело бы точно такую же синтаксическую структуру, но если только дело не происходит в волшебной стране, *a smile* является образом действия (Manner), а не инструментом. Назначение семантических ролей вскрывает такие структуры:



Помимо наблюдаемых свойств (буквы, слова, счетчики, длины, линейные расстояния, частоты и т. д.), мы можем рассматривать также производные лингвистические свойства слов, предложений и документов. Например, можно было бы рассмотреть *метку части речи* (part-of-speech tag – POS) слова внутри документа (является ли слово существительным, глаголом, прилагательным или детерминативом?), его *синтаксическую роль* (например, во фразе «the key opened the door» *key* играет роль Instrument, тогда как во фразе «the boy opened the door» *boy* – это Agent). Если даны два слова в предложении, то можно рассмотреть *дерево синтаксических зависимостей* предложения и поддерево или путь, соединяющий эти два слова внутри дерева, а также свойства этого пути. Слова, расположенные далеко друг от друга в терминах количества разделяющих их слов, могут быть близки в такой синтаксической структуре.

Если выйти за рамки предложения, то можно рассмотреть *дискурсные отношения*, которые связывают между собой предложения, например: Elaboration (уточнение), Contradiction (противоречие), CauseEffect (причинно-следственная связь) и т. д. Эти отношения часто выражаются дискурсными маркерами – словами типа *moreover* (кроме того), *however* (однако) и *and* (и), – но могут выражаться и не столь очевидными сигналами.

[1] Мальчик открыл дверь с улыбкой. – *Прим. перев.*

Еще одно важное явление – *анафора*. Рассмотрим последовательность предложений *the boy opened the door with a key. It*$_1$ *wasn't locked and he*$_1$ *entered the room. He*$_2$ *saw a man. He*$_3$ *was smiling*[1]. Разрешение анафоры (называемое также *разрешением кореференции*) скажет нам, что It_1 относится к двери (а не к ключу и не к мальчику), he_2 относится к мальчику, а he_3, вероятно, относится к мужчине.

Метки частей речи, синтаксические роли, дискурсные отношения, анафора и т. д. – все это концепции, основанные на лингвистических теориях, которые разрабатывались лингвистами в течение долгого времени с целью выявления правил и закономерностей в хаотичной системе человеческого языка. Хотя многие аспекты правил, управляющих языком, – все еще предмет споров, а другие могут показаться излишне жесткими или упрощенными, изложенные выше (и другие) концепции действительно улавливают широкий и важный массив обобщений и закономерностей языка.

Нужны ли лингвистические концепции? Некоторые поборники машинного обучения заявляют, что такие выведенные вручную лингвистические свойства не нужны и что нейронная сеть сможет сама обучиться подобным промежуточным представлениям (или эквивалентным им, или даже лучшим). Так это или не так, пока не ясно. Лично я считаю, что многие лингвистические концепции действительно могут быть выведены сетью самостоятельно, если снабдить ее достаточным объемом данных и подтолкнуть в правильном направлении[2]. Однако во многих случаях для решения интересующей нас задачи не хватает обучающих данных, и тогда было бы очень ценно снабдить сеть явными общими концепциями. И даже если данных достаточно, иногда требуется направить сеть на определенные аспекты текста и подсказать, что другие аспекты следует игнорировать. Для этого уместно включить общие концепции в дополнение или даже вместо лежащих на поверхности форм слов. Наконец, даже если не использовать лингвистические свойства в качестве входных признаков, все равно может быть полезно направить сеть в нужную сторону, используя их в роли дополнительного учителя в многозадачной конфигурации обучения (*см.* главу 20) или путем проектирования сетевой архитектуры либо парадигмы обучения, более подходящей для обучения конкретному лингвистическому явлению. В общем и целом есть достаточно свидетельств в пользу того, что лингвистические концепции позволяют лучше понять язык и повысить качество производственных систем.

Для дальнейшего чтения. При работе с текстом на естественном языке очень полезно знать о лингвистических концепциях, стоящих за буквами и словами, а также о существующих вычислительных инструментах и доступных ресурсах. Эту тему мы едва затронули. В книге Bender [2013] приведен хороший краткий обзор лингвистических концепций, ориентированный на читателей с подготовкой в области компьютеров. Обсуждение современных методов, инструментов и ресурсов NLP *см.* в книге Jurafsky and Martin [2008], а также в различных специализированных изданиях, вышедших в этой серии[3].

---

[1] Мальчик открыл дверь ключом. Она не была заперта, и он вошел в комнату. Он увидел мужчину. Он улыбался. – *Прим. перев.*

[2] *См.*, например, эксперимент, описанный в разделе 16.1.2, в котором нейронная сеть обучается концепции согласования субъекта и глагола в английском языке и попутно выводит концепции существительного, глагола, грамматического числа и некоторые иерархические лингвистические структуры.

[3] Структуры, связанные с синтаксической зависимостью, обсуждаются в работе Kübler et al. [2008], а семантические роли – в работе Palmer et al. [2010].

6.2.3. Базовые и комбинационные признаки

Во многих случаях нас интересует конъюнкция признаков, встречающихся вместе. Например, знание двух индикаторов «слово *book* встретилось в окне» и «часть речи Verb (глагол) встретилась в окне», безусловно, менее информативно, чем знание того, что «слово *book*, которому назначена часть речи Verb, встретилось в окне». Аналогично, если мы сопоставляем каждому индикаторному признаку весовой параметр (как в линейных моделях), то знание того, что имеют место два раздельных признака: «в позиции –1 находится слово *like*» и «в позиции –2 находится слово *not*», – почти бесполезно в сравнении с очень индикативным комбинированным признаком «в позиции –1 находится слово *like* и в позиции –2 находится слово *not*». Точно так же знание о том, что документ содержит слово *Paris* (Париж), свидетельствует в пользу того, что этот документ относится к категории Travel (Путешествия), и то же самое можно сказать о слове *Hilton*[1]. Однако наличие в документе обоих слов свидетельствует уже не о категории Travel, а о категории Celebrity (Знаменитости) или Gossip (Светская жизнь)[2].

В линейных моделях невозможно назначить конъюнкции событий (произошло X, и произошло Y, и …) оценку, отличную от суммы оценок каждого события в отдельности, если только конъюнкция не моделируется как самостоятельный признак. Таким образом, при проектировании признаков для линейной модели мы должны определять не только *базовые*, но и многие *комбинационные признаки*[3]. Множество возможных комбинаций очень велико, и для построения информативного и вместе с тем относительно компактного множества необходим опыт человека, и все равно без проб и ошибок не обойтись. На самом деле в проектирование решений вида «включить признаки словоформы X в позиции –1 и словоформы Y в позиции +1, но не включать признаки словоформы X в позиции –2 и словоформы Y в позиции –1» вложено очень много усилий.

6.2.4. N-граммные признаки

Частным случаем комбинационных признаков являются *n-граммы* – последовательности соседних слов заданной длины. Мы уже встречались с буквенными биграммами в задаче распознавания языка (глава 2). На практике часто используются также словесные биграммы и *триграммы* (последовательности трех элементов), составленные из букв или слов. Иногда применяются четырехграммы и пятиграммы, составленные из букв, но очень редко из слов – из-за проблем разреженности. Интуитивно должно быть понятно, почему словесные биграммы более информативны, чем отдельные слова: они улавливают такие конструкции, как *New York*, *not good* и *Paris Hilton*. И действительно, представление в виде мешка биграмм гораздо мощнее, чем в виде мешка слов, и во многих ситуациях у него просто нет соперников. Разумеется, не все биграммы одинаково информативны; такие биграммы, как *of the*, *on a*, *the boy* и т. д., встречаются очень часто и для

---

[1] Международная сеть отелей. – *Прим. перев.*

[2] Пэрис Хилтон – известная «светская львица». – *Прим. перев.*

[3] Это прямое проявление проблемы XOR, обсуждавшейся в главе 3, а вручную определенные комбинационные признаки – это функция ϕ, отображающая линейно неразделимые векторы базовых признаков в пространство более высокой размерности, где данные с большей вероятностью можно разделить с помощью линейной модели.

большинства задач не более информативны, чем составляющие их слова. Однако заранее очень трудно сказать, какие *n*-граммы окажутся полезны в данной задаче. Стандартное решение – включать все *n*-граммы не длиннее заданного порога и позволить схеме регуляризации модели отбросить менее интересные, назначив им очень малые веса.

Отметим, что классические архитектуры нейронных сетей, например МСП, в общем случае *не могут* самостоятельно вывести *n*-граммные признаки из документа: многослойный перцептрон, которому на вход подается вектор признаков документа типа мешка слов, мог бы обучиться комбинациям вида «слово X встречается в документе, и слово Y встречается в документе», но не признаку «биграмма XY встречается в документе». Таким образом, *n*-граммные признаки полезны также в контексте нелинейной классификации.

Многослойный перцептрон *может* вывести *n*-граммы, если применяется к окнам фиксированного размера с позиционной информацией: комбинация «в позиции –1 находится слово X» и «в позиции –2 находится слово Y» – это, по существу, биграмма XY. Но для поиска информативных для данной задачи *n*-граммных признаков на основе последовательностей слов переменной длины проектируются специализированные архитектуры, в частности сверточные сети (глава 13). Двунаправленные РНС (главы 14 и 16) являются дальнейшим обобщением концепции *n*-грамм, они чувствительны как к информативным *n*-граммам переменной длины, так и к *n*-граммам с внутренними пропусками.

6.2.5. Дистрибутивные признаки

До сих пор мы относились к словам как к дискретным, не связанным между собой символам: с точки зрения алгоритма, слова *pizza*, *burger* и *chair* одинаково похожи (или не похожи) друг на друга.

Мы все же пришли к некоторой форме обобщения на уровне типов слов путем отображения их в крупные категории, например части речи или синтаксические роли («*the*, *a*, *an*, *some* – детерминитивы»). Мы сумели свести флективные формы слов к леммам («*book*, *booking*, *booked* имеют общую лемму *book*»). Мы рассмотрели принадлежность слов к спискам или словарям («*John*, *Jack* и *Ralph* встречаются в списке распространенных в США имен») и изучили связи слов с другими словами, зафиксированными в лексических ресурсах типа WordNet. Однако все эти решения весьма ограничены: они либо дают слишком грубые различия, либо полагаются на специальные вручную составленные словари. Не имея списка продуктов питания, мы никогда не узнаем, что слово *pizza* больше похоже на *burger*, чем на *chair*, а еще труднее будет узнать, что *pizza* больше похожа на *burger*, чем на *icecream* (мороженое).

Дистрибутивная гипотеза языка, выдвинутая в работах Firth [1957] и Harris [1954], утверждает, что смысл слова можно вывести из контекста, в котором оно используется. Наблюдая совместную встречаемость комбинаций слов в большом корпусе текстов, мы можем установить, что контексты, в которых встречается слово *burger*, очень похожи на те, в которых встречается слово *pizza*, менее похожи на те, где встречается слово *icecream*, и уж совсем не похожи на те, где встречается слово *chair*. За прошедшие годы было разработано много алгоритмов, в которых используется это свойство и которые обучаются обобщению слов на основе контекстов, где они встречаются. Эти алгоритмы можно отнести к двум широким

категориям: *методы на основе кластеризации*, которые относят похожие слова к одному кластеру и представляют каждое слово его принадлежностью к кластеру [Brown et al., 1992, Miller et al., 2004], и *методы на основе погружения*, которые представляют слово в виде вектора, так что похожим словам (со схожим распределением) соответствуют похожие векторы [Collobert and Weston, 2008, Mikolov et al., 2013b]. В работе Turian et al. [2010] приведено обсуждение и сравнение этих подходов.

Эти алгоритмы вскрывают многие аспекты схожести между словами и могут использоваться для вывода хороших признаков слов: например, можно заменять каждое слово идентификатором его кластера (тогда оба слова *June* и *aug* будут заменены идентификатором cluster732), а редкие или не встречавшиеся при обучении слова заменять распространенным наиболее похожим словом или просто брать сам вектор слова в качестве его представления.

Однако использовать подобную информацию о схожести слов нужно осторожно, поскольку возможны непредвиденные последствия. Например, в некоторых приложениях очень полезно считать Лондон и Берлин похожими словами, тогда как в других (скажем, при заказе авиабилетов или при переводе документа) различие между ними критично.

Подробнее методы погружения слов и использование векторов слов рассматриваются в главах 10 и 11.

Глава 7

Примеры признаков в NLP

Обсудив различные доступные источники информации для вывода признаков из текста на естественном языке, перейдем к изучению конкретных задач классификации в NLP и подходящих для них признаков. Хотя нейронные сети обещают значительно уменьшить потребность в ручном конструировании признаков, нам все равно следует принимать эти источники информации во внимание при проектировании моделей: мы хотим, чтобы сеть могла эффективно использовать все доступные сигналы, а для этого можно либо дать к ним прямой доступ, применив конструирование признаков и спроектировав сеть так, чтобы она раскрывала необходимые сигналы, либо добавив их в виде дополнительных сигналов потери на этапе обучения моделей[1].

7.1. Классификация документов: определение языка

В задаче определения языка дан документ или предложение и требуется отнести его к одному языку из фиксированного набора. В главе 2 мы видели, что для этой задачи очень полезным представлением признаков является *мешок буквенных биграмм*. Точнее, каждая возможная буквенная биграмма (или каждая буквенная биграмма, встречающаяся, по меньшей мере, *k* раз хотя бы в одном языке) является базовым признаком, а значение базового признака для данного документа равно счетчику этого признака в документе.

Аналогично ставится задача об *определении кодировки*. В данном случае хорошим представлением признаков является мешок байтовых биграмм.

---

[1] Кроме того, линейные или логлинейные модели с вручную спроектированными признаками по-прежнему очень эффективны при решении многих задач. Они вполне конкурентоспособны с точки зрения верности, а также очень легко обучаются и развертываются даже в крупномасштабных системах. К тому же о них проще рассуждать и их легче отлаживать, чем нейронные сети. И уж во всяком случае такие модели можно рассматривать как эталон для сравнения с проектируемой вами сетью.

7.2. Классификация документов: тематическая классификация

В задаче тематической классификации дан документ и требуется отнести его к одной теме из заранее заданного набора (например, экономика, политика, спорт, досуг, светская хроника, стиль жизни, прочее).

Здесь уровень букв не слишком информативен, так что базовыми единицами будут слова. В этой задаче порядок слов не особенно важен (за исключением, быть может, пар соседних слов, как в биграммах). Поэтому хорошим набором признаков будет *мешок слов*, возможно, в совокупности с мешком словесных биграмм (каждое слово и каждая словесная биграмма – базовые признаки).

Если обучающих примеров не слишком много, то имеет смысл предварительно обработать документ, заменив каждое слово леммой. Можно также заменить или дополнить слова дистрибутивными признаками, например кластерами слов или векторами погружения слов.

При использовании линейного классификатора желательно также рассматривать все пары слов (необязательно соседних), встречающиеся в одном и том же документе, в качестве базовых признаков. Это даст огромное множество потенциальных базовых признаков, которое придется сократить, придумав какую-то эвристику, например учитывать только пары слов, встречающиеся в заданном числе документов. Для нелинейных классификаторов такой необходимости не возникает.

Если используется мешок слов, то иногда полезно взвешивать каждое слово пропорционально его информативности, например применяя схему весов TF-IDF (раздел 6.2.1). Однако зачастую алгоритм обучения способен самостоятельно назначать веса. Еще один вариант – использовать не счетчики слов, а индикаторы: каждое слово в документе (или каждое слово сверх определенного количества) будет представлено только единожды, независимо от того, сколько раз оно встречается в документе.

7.3. Классификация документов: установление авторства

В задаче установления авторства [Koppel et al., 2009] дан текст и требуется установить личность автора (выбрав его из фиксированного числа возможных авторов) или другие характеристики автора, например пол, возраст, родной язык.

Характер информации, необходимой для решения этой задачи, сильно отличается от той, что необходима для тематической классификации, – зацепки очень тонкие, они относятся к стилистическим свойствам текста, а не к составляющим его полнозначным словам.

Поэтому при выборе признаков мы должны держаться подальше от полнозначных слов и сконцентрироваться на стилистических особенностях[1]. Для таких за-

[1] Можно было бы возразить, что для определения возраста или пола имеет смысл обращать внимание также на слова, поскольку существует сильная корреляция между полом и возрастом человека и темами, на которые он пишет, а также регистром букв. Вообще

дач стоит обратить внимание на *метки частей речи* (POS) и *неполнозначные слова*. К таковым относятся слова типа *on*, *of*, *the*, *and*, *before* и т. п., которые не несут особого смысла сами по себе, а связываются с полнозначными словами и придают смысл их сочетаниям; к той же группе относятся местоимения (*he*, *she*, *I*, *they* и т. д.). Хорошей аппроксимацией неполнозначных слов может служить список 300 (или около того) самых часто встречающихся слов в большом корпусе. Сосредоточившись на таких признаках, мы сможем обучиться улавливать тонкие особенности авторского стиля, которые очень трудно подделать.

Хороший набор признаков для задачи установления авторства включает мешок неполнозначных слов и местоимений и мешки частеречных биграмм, триграмм и четырехграмм. Кроме того, имеет смысл рассмотреть *плотность* неполнозначных слов (т. е. отношение количества неполнозначных слов к количеству полнозначных слов в окне текста), мешок биграмм неполнозначных слов после удаления полнозначных и распределение расстояний между соседними неполнозначными словами.

7.4. Слово в контексте: частеречная разметка

В задаче частеречной разметки дано предложение и требуется правильно сопоставить каждому слову часть речи. Части речи берутся из предопределенного набора, в данном случае мы возьмем набор из проекта *Universal Treebank Project* [McDonald et al., 2013, Nivre et al., 2015], содержащий 17 меток[1].

Частеречная разметка обычно моделируется как структурная задача – метка первого слова может зависеть от метки третьего, но ее можно довольно хорошо аппроксимировать, если классифицировать каждое слово по отдельности, используя окно, содержащее по два слова с каждой стороны. Если слова помечаются в определенном порядке, например слева направо, то можно также обусловить каждое предсказание метки предсказаниями предыдущих меток. В момент классификации слова w_i наша функция выделения признаков будет иметь доступ ко всем словам предложения (и составляющим их буквам), а также к принятым ранее решениям о разметке (т. е. меткам, которые были назначены словам w_1, …, w_{i-1}). Здесь мы говорим о признаках, как если бы они использовались в изолированной задаче классификации. В главе 19 мы обсудим структурное обучение с использованием того же набора признаков.

Источники информации для задачи частеречной разметки можно разделить на внутренние (основанные на самом слове) и внешние ключи (основанные на контексте слова). К внутренним ключам относятся: идентификатор слова (например, некоторые слова более вероятны в роли существительных, чем другие), префиксы, суффиксы, орфографическая форма слова (в английском языке слова,

говоря, это правда, но если нас интересует криминалистическая экспертиза или враждебное поведение, при котором автор намерен скрыть свой пол или возраст, то лучше не полагаться на контентные признаки, поскольку их гораздо легче подделать, чем более тонкие стилистические особенности.

[1] Прилагательное, прилог (предлог или послелог), наречие, вспомогательный глагол, соединительный союз, детерминатив, междометие, существительное, числительное, частица, местоимение, имя собственное, знак препинания, подчинительный союз, символ, глагол, прочее.

оканчивающиеся на -ed, скорее всего, являются глаголами в прошедшем времени, слова, начинающиеся с un-, – с большой вероятностью прилагательные, а слова, начинающиеся заглавной буквой, – наверное, имена собственные) и частота слова в большом корпусе (так, редкие слова, скорее всего, являются существительными). К внешним ключам относятся идентификаторы, префиксы и суффиксы слов, окружающих данное, а также предсказанные части речи для предшествующих слов.

ПЕРЕКРЫВАЮЩИЕСЯ ПРИЗНАКИ Если в качестве признака берется словоформа, то зачем нужны префиксы и суффиксы? Ведь все они – детерминированные функции слова. Причина в том, что если встретится слово, которое не предъявлялось на этапе обучения (несловарное, или *OOV-слово*), или слово, которое встречалось всего несколько раз (*редкое слово*), то у нас нет достаточно надежной информации для принятия решения. В таком случае разумно обратиться к префиксам и суффиксам, которые могут дать полезные подсказки. Включая в качестве признаков префиксы и суффиксы слов, часто встречавшихся во время обучения, мы даем алгоритмам возможность лучше подстраивать веса и надеяться на то, что они сумеют воспользоваться этой информацией, встретив несловарное слово.

Ниже приведен пример хорошего набора базовых признаков для частеречной разметки:

- слово = X;
- 2-буквенный-суффикс = X;
- 3-буквенный-суффикс = X;
- 2-буквенный-префикс = X;
- 3-буквенный-префикс = X;
- слово-составлено-из-заглавных-букв;
- слово-содержит-дефис;
- слово-содержит-цифру;
- for P in [−2, −1, +1, +2]:
  - слово в позиции P = X;
  - 2-буквенный-суффикс слова в позиции P = X;
  - 3-буквенный-суффикс слова в позиции P = X;
  - 2-буквенный-префикс слова в позиции P = X;
  - 3-буквенный-префикс слова в позиции P =X;
  - слово в позиции P = X составлено из заглавных букв;
  - слово в позиции P = X содержит дефис;
  - слово в позиции P = X содержит цифру;
- часть речи, предсказанная для слова в позиции −1 = X;
- часть речи, предсказанная для слова в позиции −2 = X.

Кроме того, может быть полезна дистрибутивная информация, например кластеры слов или векторы погружения слов, особенно для слов, которых не было в обучающем корпусе, поскольку слова с похожими метками части речи имеют тенденцию совместно встречаться в более похожих контекстах, чем слова с другими метками части речи.

7.5. Слово в контексте: распознавание именованных сущностей

В задаче распознавания именованных сущностей (named-entity recognition – NER) дан документ и требуется найти в нем именованные сущности, например *Milan*, *John Smith*, *McCormik Industries*, *Paris*, а также сопоставить каждой одну из предопределенных категорий, например: Location (местоположение), Organization (организация), Person (физическое лицо) или Other (прочие). Заметим, что это контекстно-зависимая задача, поскольку Milan может быть как местоположением (город), так и организацией (футбольная команда, «Вечером Милан играл с Барсой»), а Paris – как названием города, так и именем человека.

Типичные входные данные для этой задачи имеют вид:

John Smith, president of McCormik Industries visited his niece Paris in Milan, reporters say,

а ожидаемый результат выглядит так:

[$_{\text{PER}}$ John Smith], president of [$_{\text{ORG}}$ McCormik Industries] visited his niece [$_{\text{PER}}$ Paris] in [$_{\text{LOC}}$ Milan], reporters say.

Хотя NER является *задачей сегментации предложения* – помеченные скобки окружают непересекающиеся участки предложения, – часто она моделируется как задача разметки предложения, подобно частеречной разметке. Применение разметки для решения задачи сегментации осуществляется с помощью *BIO-меток*[1]. Каждому слову сопоставляется одна из меток, как показано в табл. 7.1.

Таблица 7.1. BIO-метки для распознавания именованных сущностей

Метка	Назначение
O	Не является частью именованной сущности
B-PER I-PER	Первое слово имени человека Продолжение имени человека
B-LOC I-LOC	Первое слово названия местоположения Продолжение названия местоположения
B-ORG I-ORG	Первое слово названия организации Продолжение названия организации
B-MISC I-MISC	Первое слово другой именованной сущности Продолжение другой именованной сущности

Приведенное выше предложение было бы размечено так:

John/B-PER Smith/I-PER,/O president/O of/O McCormik/B-ORG Industries/I-ORG visited/O his/O niece/O Paris/B-PER in/O Milan/B-LOC,/O reporters/O say/O. /O.

Перевод неперекрывающихся сегментов в BIO-метки и обратно тривиален.

Как и частеречная разметка, задача NER структурная, так как решения о разметке для разных слов перекрываются (пребывание в пределах одной сущности

[1] В литературе изучены различные варианты схем BIO-разметки, некоторые работают лучше описанного. *См*. Lampleet al. [2016], Ratinov and Roth [2009].

вероятнее переключения, т. е. последовательность «John Smith Inc.» с большей вероятностью будет размечена как B-ORG I-ORG I-ORG, чем как B-PER I-PER B-ORG). Однако и на этот раз решение можно достаточно хорошо аппроксимировать, принимая решения о классификации независимо.

Базовый набор признаков для задачи NER примерно такой же, как для частеречной разметки, он опирается на слова в окне шириной 2 слова по обе стороны от фокусного слова. Признаки, применяемые для частеречной разметки, полезны и для NER (например, суффикс -ville характерен для местоположений, а префикс Mc- – для фамилий), но помимо них имеет смысл рассмотреть идентификаторы слов, окружающих *другие* вхождения данного слова в текст, а также индикаторные функции, проверяющие, находится ли слово в заранее составленном списке лиц, местоположений и организаций. Дистрибутивные признаки, такие как кластеры слов или векторы слов, также чрезвычайно полезны в задаче NER. Подробное обсуждение признаков для NER *см*. в работе Ratinov and Roth [2009].

7.6. Слово в контексте, лингвистические признаки: разрешение лексической многозначности предлогов

Предлоги, т. е. слова *on*, *in*, *with*, *for* и т. п., служат для связи предикатов с их аргументами и существительных с их предложными модификаторами. Предлоги встречаются очень часто и при этом крайне неоднозначны[1]. Рассмотрим, например, слово *for* в следующих предложениях.

(1) a. We went there *for* lunch (Мы пошли на обед).
b. He paid *for* me (Он заплатил за меня).
c. We ate *for* two hours (Мы ели в течение двух часов).
d. He would have left *for* home, but it started raining (Он пошел бы домой, но начался дождь).

Слово *for* в каждом из них играет разные роли: в (a) оно означает Purpose (назначение), (b) a Beneficiary (бенефициара), в (c) Duration (продолжительность), а в (d) Location (местоположение).

Чтобы полностью понять смысл предложения, необходимо правильно понимать смысл встречающихся в нем предлогов. Задача *разрешения лексической многозначности предлогов* заключается в присваивании правильного смысла предлогам в зависимости от контекста, причем реестр смыслов конечен. В работе Schneider et al. [2015, 2016] обсуждается сама задача, предлагается унифицированный реестр смыслов, охватывающий многие предлоги, а также небольшой аннотированный корпус предложений из онлайновых обзоров, содержащий 4250 упоминаний предлогов, каждое из которых аннотировано смыслом предлога[2].

---

[1] К английскому языку это относится в гораздо большей степени, чем к русскому. – *Прим. перев.*

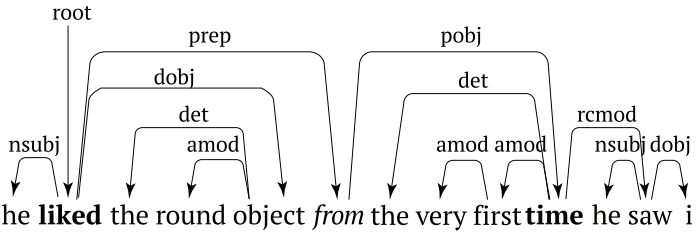
[2] Доступны также более ранние реестры и аннотированные корпусы текстов для этой задачи. *См.*, например, Litkowski and Hargraves [2005, 2007], Srikumar and Roth [2013a].

Что считать хорошим набором признаков для задачи разрешения лексической многозначности предлогов? Мы возьмем набор признаков, подсказанный работой Hovy et al. [2010].

Очевидно, что полезным признаком является сам предлог (например, распределение возможных смыслов предлога *in* сильно отличается от распределения смыслов для *with* или *about*). Помимо этого, мы будем рассматривать контекст слова. Но фиксированное окно вокруг предлога не идеально с точки зрения информационного содержания. Рассмотрим, к примеру, следующие предложения.

(2) a. He liked the round object *from* the very first time he saw it (Ему понравился круглый предмет *с* того самого момента, как он его впервые увидел).
b. He saved the round object *from* him the very first time they saw it (Он спас круглый предмет *от* него в первый же раз, как увидел его).

Эти два употребления *from* различаются по смыслу, но многие слова в окне, окружающем слово, либо не информативные, либо даже вводят в заблуждение. Нам необходим более подходящий механизм выбора информативных контекстов. Один из вариантов – воспользоваться эвристикой, например «первый глагол слева» и «первое существительное справа». Так мы найдем тройки ⟨liked, from, time⟩ и ⟨saved, from, him⟩, которые действительно улавливают основной смысл предлога. Лингвисты говорят, что эта эвристика помогает найти *субъект* и *объект* предлога. Зная идентификатор предлога, а также субъект и объекты, человек во многих случаях может вывести его смысл, прибегая к рассуждениям о детальной семантике слов. Описанная эвристика для извлечения объекта и субъекта требует использования частеречного разметчика для идентификации существительных и глаголов. Она довольно хрупкая – нетрудно представить случаи, когда она не дает правильного результата. Можно было бы усовершенствовать эвристику, добавив новые правила, но более надежен подход, основанный на анализаторе зависимостей: субъект и объект легко можно получить из синтаксического дерева, что устраняет необходимость в сложной эвристике:



Конечно, анализатор, применяемый для построения дерева, тоже может ошибаться. Для большей надежности можно рассмотреть субъект и объект, полученные от анализатора, *и* субъект и объект, полученные с помощью эвристики, и использовать в качестве признаков все четыре (т. е. parse_gov = X, parse_obj = Y, heur_gov = Z, heur_obj = W), поручив процессу обучения решить, какой источник надежнее и как их сбалансировать.

После извлечения субъекта и объекта (и, быть может, соседних с ними слов) мы можем использовать их как основу для дальнейшего выделения признаков. Для каждого элемента можно было бы извлечь следующую информацию:

- фактическую *поверхностную форму* слова;
- *лемму* слова;
- *часть речи* слова;
- *префиксы* и *суффиксы* слова (указывающие на прилагательные степени, количества, порядка и т. д., например: *ultra-*, *poly-*, *post-*, – а также на некоторые различия между агентивными и неагентивными глаголами);
- *кластер слова* или *дистрибутивный вектор* слова.

Если мы допускаем использование внешних лексических ресурсов и неособенно возражаем против увеличения пространства признаков, то в работе Hovy et al. [2010] описан способ использования *признаков, основанных на WordNet*. Для каждого субъекта и объекта мы могли бы получить из WordNet много индикаторов, например:

- имеется ли в WordNet соответствующая этом слову статья;
- гиперонимы первого синсета слова;
- гиперонимы всех синсетов слова;
- синонимы для первого синсета слова;
- синонимы для всех синсетов слова;
- все термы в определении слова;
- суперсмысл слова (суперсмыслы, называемые также файлами лексикографа, в терминологии WordNet – относительно высокие уровни иерархии, включающие такие концепты, как животное, быть частью тела, быть эмоцией, быть пищей и т. д.);
- различные дополнительные индикаторы.

Этот процесс может дать десятки, а иногда и более сотни базовых признаков для каждого употребления предлога, например: hyper_1st_syn_gov = a, hyper_all_syn_gov = a, hyper_all_syn_gov = b, hyper_all_syn_gov = c, …, hyper_1st_syn_obj = x, hyper_all_syn_obj = y, …, term_in_def_gov = q, term_in_def_gov = w и т. д.

Детали *см*. в работе Hovy et al. [2010].

Задача разрешения лексической неоднозначности предлогов – пример высокоуровневой проблемы семантической классификации, нуждающейся в наборе признаков, который невозможно легко вывести из поверхностных форм. Для ее решения полезна лингвистическая предобработка (т. е. частеречная разметка и синтаксический анализ), а также получение информации из поддерживаемых вручную семантических лексиконов.

7.7. Отношения между словами в контексте: анализ методом разложения на дуги

В задаче *анализа зависимостей* дано предложение и требуется вернуть для него *синтаксическое дерево зависимостей*, показанное на рис. 7.1. Каждому слову, кроме главного слова предложения, сопоставляется родительское слово, а для главного слова родителем является специальный символ *ROOT*.

Дополнительные сведения о задаче анализа зависимостей, ее лингвистических основаниях и подходах к решению *см*. в книге Kübler et al. [2008].

Одним из подходов к моделированию этой задачи является разложение на дуги [McDonald et al., 2005], когда каждому из n^2 возможных отношений слово-слово

(дуг) назначается независимая от остальных оценка, после чего мы ищем допустимое дерево с максимальной полной оценкой. Назначение оценок производит обученная функция оценивания ArcScore(*h*, *m*, *sent*), которая получает предложение и индексы *h* и *m* двух входящих в него слов, которые считаются кандидатами на соединение (*h* – индекс заглавного слова-кандидата, а *m* – индекс кандидата на роль модификатора). Обучение функции оценивания таким образом, чтобы она хорошо работала совместно с процедурой поиска, обсуждается в главе 19. Здесь же мы сосредоточимся на признаках, используемых в функции оценивания.

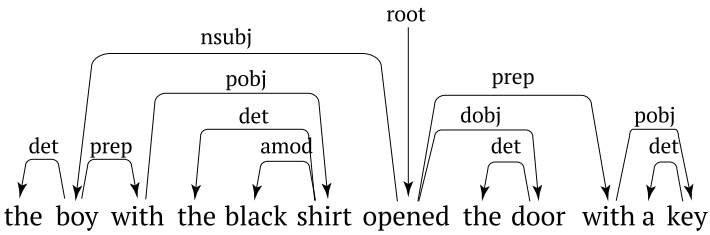


Рис. 7.1 ❖ Дерево зависимостей

Пусть имеется предложение из *n* слов $w_{1:n}$, которым соответствуют части речи $p_{1:n}$: $sent = (w_1, w_2, \ldots, w_n, p_1, p_2, \ldots, p_n)$. При рассмотрении дуги между словами w_h и w_m мы можем воспользоваться описанной ниже информацией.

Начнем с обычных подозреваемых:

- словоформа (и метка части речи) *заглавного слова*;
- словоформа (и метка части речи) *слова-модификатора* (некоторые слова вряд ли могут быть заглавными или модификаторами, чем бы они ни были связаны. Например, детерминативы («the», «a») часто являются модификаторами, но никогда – заглавными словами);
- слова (и метки частей речи) в окне шириной по два слова с каждой стороны от заглавного слова, включая их относительные позиции;
- слова (и метки частей речи) в окне шириной по два слова с каждой стороны от слова-модификатора, включая их относительные позиции (информация об окне нужна, чтобы у слова был какой-то контекст. Слова ведут себя поразному в разных контекстах).

Мы используем не только сами словоформы, но и метки частей речи. Словоформы дают очень конкретную информацию (например, что *cake* (пирожное) – подходящий кандидат на роль объекта для *ate* (съел)), тогда как части речи – низкоуровневую синтаксическую информацию, в большей степени допускающую обобщение (например, что детерминативы и прилагательные – хорошие модификаторы для существительных, а существительные – хорошие модификаторы для глаголов). Поскольку размер обучающего корпуса для деревьев зависимостей обычно довольно ограничен, имеет смысл дополнить или заменить слова, воспользовавшись дистрибутивной информацией в виде кластеров слов или предобученных погружений слов, которые улавливают общие черты похожих слов, а заодно компенсируют недостаточное присутствие слов в составе обучающих данных.

Мы не рассматриваем префиксы и суффиксы слов, поскольку к задаче грамматического разбора они не имеют прямого отношения. Конечно, аффиксы несут

важную синтаксическую информацию (является ли слово существительным? или глаголом в прошедшем времени?), но эта информация уже доступна нам в виде меток частей речи. Если бы мы производили разбор, не располагая частеречными признаками (например, если бы анализатор отвечал и за разбор, и за частеречную разметку), то было бы разумно включить и информацию о суффиксах.

Разумеется, при использовании линейного классификатора надо позаботиться также о *комбинациях признаков* такого вида: «кандидат на роль заглавного слова X, и кандидат на роль слова-модификатора Y, и метка части речи заглавного слова Z, и слово перед словом-модификатором W». На самом деле анализаторы зависимостей, основанные на линейных моделях, насчитывают сотни таких комбинационных признаков.

Помимо этих обычных подозреваемых, информативными являются следующие признаки.

- *Расстояние* между словами w_h и w_m в предложении, $dist = |h - m|$. Для некоторых расстояний вероятность занять почетное место в отношении зависимости больше, чем для других.
- *Направление* между словами. Имея в виду английский язык, предположим, что w_m – детерминатив («the»), а w_h – существительное («boy»). Тогда вполне вероятно, что между ними существует дуга, если $m < h$, и совершенно невероятно, что такая дуга существует при $m > h$.
- Все слова (и их метки частей речи), расположенные *между* головой и модификатором в предложении. Эта информация полезна для разрешения потенциальных конфликтов отнесения. Например, детерминатив в позиции w_m может модифицировать существительное в позиции $w_{h>m}$, но не в том случае, когда слово в позиции w_k ($m < k < h$) между ними тоже является детерминативом. Заметим, что количество слов между головой и модификатором теоретически не ограничено (и к тому же меняется от примера к примеру), поэтому необходим способ кодирования переменного числа признаков, что наводит на мысль о мешке слов.

Глава 8

От текстовых признаков к входным данным

В главах 2 и 4 мы обсудили классификаторы, принимающие на входе векторы признаков, не вдаваясь в детали содержимого этих векторов. В главах 6 и 7 мы рассмотрели источники информации, которые могут послужить для формирования базовых признаков в различных задачах NLP. В этой главе мы поговорим о том, как перейти от списка базовых признаков к вектору признаков, который может быть подан на вход классификатору.

Напомним, что в главах 2 и 4 мы представили машинно-обучаемые модели (линейные, логлинейные и многослойный перцептрон). Модели – это параметрические функции $f(\boldsymbol{x})$, которые принимают d_{in}-мерный вектор $\boldsymbol{x}$ и порождают d_{out}-мерный выходной вектор. Эту функцию часто называют *классификатором*, который сопоставляет входу $\boldsymbol{x}$ степень принадлежности к одному или более из d_{out} классов. Функция может быть как простой (в случае линейной модели), так и более сложной (в случае произвольной нейронной сети). В этой главе нас будет интересовать вход $\boldsymbol{x}$.

8.1. Кодирование категориальных признаков

В задачах обработки естественного языка большинство признаков – дискретные, категориальные величины: слова, буквы или метки частей речи. Как закодировать такие категориальные данные, чтобы их было удобно использовать в статистическом классификаторе? Мы рассмотрим два варианта: *унитарное кодирование* и *плотные векторы погружения*, а также возникающие здесь компромиссы и отношения.

8.1.1. Унитарное кодирование

В линейных и логлинейных моделях вида $f(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}$ принято рассуждать в терминах индикаторных функций и отводить под каждый возможный признак отдельное измерение. Например, при использовании в качестве представления мешка слов со словарем из 40 000 элементов $\boldsymbol{x}$ будет 40 000-мерным вектором, в котором измерение 23 227 (к примеру) соответствует слову *dog*, а измерение 12 425 – слову *cat*. Документ, содержащий 20 слов, будет представлен очень разреженным 40 000-мерным вектором, в котором отлично от нуля не более 20 элементов. Следовательно, в матрице $\boldsymbol{W}$ будет 40 000 строк, каждая из которых соответствует одному словарному слову. Если базовыми признаками являются слова

в окне шириной 5 слов вокруг целевого слова (по 2 слова с каждой стороны) с позиционной информацией, а словарь содержит 40 000 слов (т. е. признаки имеют вид word-2 = dog или word0 = sofa), то $\boldsymbol{x}$ будет 200 000-мерным вектором с 5 ненулевыми элементами, в котором измерение 19 234 (к примеру) соответствует слову word-2 = dog, а измерение 143 167 – слову word0 = sofa. Это называется *унитарной* кодировкой – каждое измерение соответствует уникальному признаку, а результирующий вектор признаков можно рассматривать как комбинацию многомерных индикаторных векторов, в которых ровно один элемент равен 1, а все остальные – 0.

8.1.2. Плотные погружения (погружения признаков)

Быть может, самым важным концептуальным скачком при переходе от линейных моделей с разреженным входом к более глубоким нелинейным моделям является отказ от представления каждого признака отдельным измерением, как в унитарном представлении, в пользу плотных векторов признаков. Это означает, что каждый базовый признак *погружается* в d-мерное пространство и представляется вектором в этом пространстве[1]. Размерность d обычно гораздо меньше числа признаков, т. е. каждый элемент словаря на 40 000 слов (закодированных 40 000-мерными унитарными векторами) можно представить в виде 100- или 200-мерного вектора. Погружения (векторные представления базовых признаков) рассматриваются как параметры сети и обучаются как любые другие параметры функции f. На рис. 8.1 показаны оба подхода к представлению признаков.

Общая структура системы классификации в NLP, основанной на нейронной сети прямого распространения, имеет такой вид.

1. Выделить набор базовых лингвистических признаков $f_1, \ldots, f_k$, релевантных предсказанию выходного класса.
2. Для каждого представляющего интерес признака f_i найти соответствующий вектор $v(f_i)$.
3. Объединить векторы (путем конкатенации, суммирования или комбинации того и другого) во входной вектор $\boldsymbol{x}$.
4. Подать $\boldsymbol{x}$ на вход нелинейного классификатора (нейронной сети прямого распространения).

Таким образом, главное изменение во входных данных при переходе от линейного к более глубокому классификатору – переход от разреженных представлений, где под каждый признак отведено отдельное измерение, к более плотному представлению, в котором каждый признак отображается в вектор. Еще одно различие состоит в том, что нам в большинстве случаев достаточно выделить только *базовые признаки*, а не их комбинации. Ниже мы подробнее поговорим об обоих изменениях.

8.1.3. Плотные векторы и унитарные представления

В чем преимущество представления признаков в виде векторов, а не уникальных идентификаторов? Всегда ли следует представлять признаки в виде плотных векторов? Рассмотрим оба вида представлений.

---

[1] Признаки разных типов могут погружаться в различные пространства. Например, словесные признаки можно представлять в 100-мерном пространстве, а частеречные – в 20-мерном.

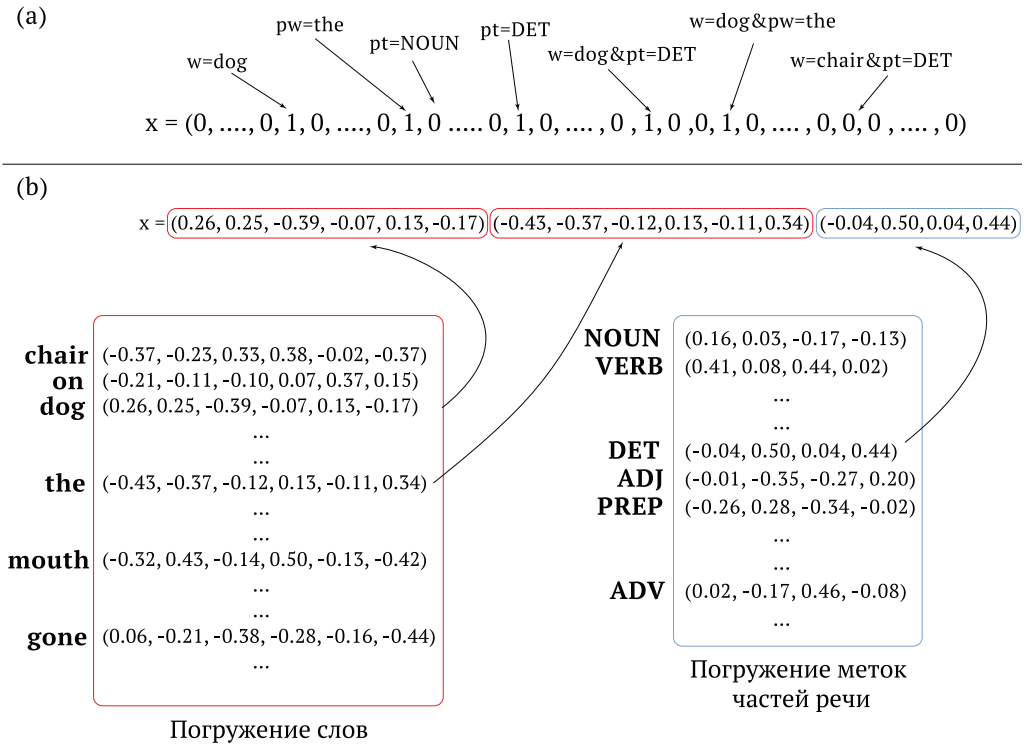


Рис. 8.1 ❖ Разреженное и плотное представление. Два способа кодирования информации: *текущее слово «dog»; предыдущее слово «the»; предыдущая метка части речи «DET»*. (a) Разреженный вектор признаков. Каждое измерение представляет один признак. Для комбинации признаков нужно отдельное измерение. Значения признаков бинарные. Размерность очень высокая. (b) Плотный, основанный на погружениях вектор признаков. Базовые признаки представлены в виде векторов. Каждому признаку соответствует несколько элементов входного вектора. Комбинации признаков явно не кодируются. Размерность низкая. Отображение признаков в векторы производится с помощью таблицы погружений

Унитарное кодирование. Под каждый признак отводится отдельное измерение.

- Размерность унитарного вектора совпадает с количеством различных признаков.
- Признаки никак не зависят друг от друга. Признак «слово равно ‘dog’» не похож на признак «слово равно ‘thinking’» ровно так же, как на признак «слово равно ‘cat’».

Плотное кодирование. Каждый признак – это d-мерный вектор.

- Размерность вектора равна d.
- В процессе обучения модели похожие признаки отображаются в похожие векторы – информация обобществляется между похожими признаками.

Одно из преимуществ использования плотных векторов малой размерности чисто вычислительное: большинство библиотек для работы с нейронными сетями не любит работать с разреженными векторами очень высокой размерности. Впро-

чем, это чисто техническое препятствие, которое можно устранить при помощи правильного проектирования.

Главное достоинство плотных представлений – способность к обобщению: если мы полагаем, что некоторые признаки могут дать схожие ключи, то стоит попытаться найти представление, способное уловить это сходство. Например, предположим, что мы много раз видели слово *dog* во время обучения, а слово *cat* всего несколько раз или вообще ни разу. Если с каждым из этих слов ассоциировано отдельное измерение, то вхождения *dog* ничего не скажут о вхождениях *cat*. Однако в представлении плотными векторами обученный вектор для *dog* может оказаться похожим на обученный вектор для *cat*, что позволит модели обобщить статистическую силу обоих событий. В этом рассуждении предполагается, что мы видели достаточно вхождений слова *cat*, так что его вектор будет похож на вектор *dog*, или что «хорошие» векторы каким-то образом переданы нам извне. Такие «хорошие» векторы слов (называемые также *предобученными погружениями*) можно получить из большого корпуса текста с помощью алгоритмов, основанных на дистрибутивной гипотезе. Подобные алгоритмы изучаются в главе 10.

В тех случаях, когда в категории сравнительно мало уникальных признаков и мы полагаем, что между разными признаками нет корреляции, можно использовать унитарное представление. Но если мы считаем, что между разными признаками в группе должны быть корреляции (например, для меток частей речи, возможно, есть основания полагать, что разные формы глагола VB и VBZ могут вести себя похоже в нашей задаче), то имеет смысл позволить сети самостоятельно выявить эти корреляции и обрести некоторую статистическую силу за счет обобществления параметров. Не исключено, что при определенных обстоятельствах – когда пространство признаков относительно мало, а обучающих данных достаточно или когда мы не хотим обобществлять статистическую информацию между разными словами, – унитарные представления могут быть выгоднее. Однако пока это открытый вопрос, убедительных свидетельств в пользу того или иного решения нет. Большая часть авторов (Chen and Manning [2014], Collobert and Weston [2008], Collobert et al. [2011]) выступает за использование плотных обучаемых векторов погружений для всех признаков. Об архитектуре нейронной сети с разреженными кодировками векторов *см*. работу Johnson and Zhang [2015].

8.2. Объединение плотных векторов

Каждому признаку соответствует плотный вектор, и разные векторы нужно как-то объединить. Наиболее популярны конкатенация, суммирование (или усреднение) и комбинации того и другого.

8.2.1. Оконные признаки

Рассмотрим кодирование окна, содержащего по k слов с каждой стороны фокусного слова в позиции i. Пусть $k = 2$, нам нужно закодировать слова в позициях $i - 2$, $i - 1$, $i + 1$ и $i + 2$. Предположим, что в окне находятся слова a, b, c и d, и обозначим $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$ и $\boldsymbol{d}$ – соответствующие векторы слов. Если нам безразличны относительные позиции слов в окне, то будем кодировать окно суммой: $\boldsymbol{a} + \boldsymbol{b} + \boldsymbol{c} + \boldsymbol{d}$. Если же относительные позиции важны, то будем использовать конкатенацию: $[\boldsymbol{a}; \boldsymbol{b}; \boldsymbol{c}; \boldsymbol{d}]$.

Хотя слову будет соответствовать один и тот же вектор независимо от его позиции в окне, позиция слова находит отражение в его позиции внутри конкатенированного вектора[1].

Возможно, нам безразличен порядок, но хотелось бы все же придавать словам, отстоящим дальше от контекстного слова, меньшую значимость, чем тем, что находятся рядом с ним. Для этого можно воспользоваться взвешенной суммой, т. е. $½\boldsymbol{a} + \boldsymbol{b} + \boldsymbol{c} + ½\boldsymbol{d}$.

Эти кодировки можно комбинировать различными способами. Допустим, что нам важно, встречается ли признак до или после контекстного слова, но расстояние от него до контекстного слова не существенно, коль скоро он находится внутри окна. Это можно закодировать с помощью комбинации суммирования и конкатенации: $[(\boldsymbol{a} + \boldsymbol{b}); (\boldsymbol{c} + \boldsymbol{d})]$.

ЗАМЕЧАНИЕ О НОТАЦИИ При описании слоев сети, которые принимают на входе конкатенированные векторы $\boldsymbol{x}$, $\boldsymbol{y}$ и $\boldsymbol{z}$, некоторые авторы явно указывают конкатенацию ($[\boldsymbol{x}; \boldsymbol{y}; \boldsymbol{z}]\boldsymbol{W} + \boldsymbol{b}$), тогда как другие применяют аффинное преобразование ($\boldsymbol{xU} + \boldsymbol{yV} + \boldsymbol{zW} + \boldsymbol{b}$). Если матрицы весов в аффинном преобразовании различаются[2], то обе нотации эквивалентны.

8.2.2. Переменное число признаков: непрерывный мешок слов

Сети прямого распространения допускают вход фиксированной размерности. Этому ограничению удовлетворяет функция, которая выделяет фиксированное количество признаков: каждый признак представляется вектором, и векторы конкатенируются. Таким образом, разные участки получающегося входного вектора соответствуют разным признакам. Но иногда количество признаков заранее неизвестно (например, в задаче классификации документов часто бывает, что каждое слово предложения является признаком). Поэтому необходимо представлять неограниченное число признаков вектором фиксированного размера. Один из способов добиться этого дает так называемый *непрерывный мешок слов* (continuous bag of words – CBOW) [Mikolov et al., 2013b]. CBOW очень похож на традиционное представление в виде мешка слов, в котором информация о порядке отбрасывается; в основе его работы лежит суммирование или усреднение векторов погружения соответствующих признаков[3]:

---

[1] Можно было бы вместо этого завести отдельное погружение для каждой пары слово–позиция, т. е. $\boldsymbol{a}^{1}$ и $\boldsymbol{a}^{-2}$ будут представлять слово a, когда оно встречается в относительных позициях +1 и −2 соответственно. При таком подходе можно использовать сумму и все-таки сохранить чувствительность к позиционной информации: $\boldsymbol{a}^{-2} + \boldsymbol{b}^{-1} + \boldsymbol{c}^{+1} + \boldsymbol{d}^{+2}$. Но информация между экземплярами слов, встречающимися в разных позициях, не обобществляется, и такое представление сложнее использовать совместно с предобученными векторами слов.

[2] Матрицы должны быть *различны* в том смысле, что изменение одной не отражается на других. Разумеется, вполне допустимо, если элементы матриц будут одинаковыми.

[3] Заметим, что если бы $v(f_i)$ были унитарными векторами, а не плотными представлениями признаков, то CBOW (уравнение (8.1)) и WCBOW (уравнение (8.2)) свелись бы к обычному (соответственно, взвешенному) представлению мешком слов, которое, в свою очередь, эквивалентно разреженному представлению векторов признаков, в котором каждый признак, являющийся бинарным индикатором, соответствует одному «слову».

$$\text{CBOW}(f_1, \ldots, f_k) = \frac{1}{k}\sum_{i=1}^{k} v(f_i). \tag{8.1}$$

Простая вариация представления CBOW – взвешенный CBOW, когда разным векторам назначаются разные веса:

$$\text{WCBOW}(f_1, \ldots, f_k) = \frac{1}{\sum_{i=1}^{k} a_i}\sum_{i=1}^{k} a_i v(f_i). \tag{8.2}$$

Здесь с каждым признаком f_i ассоциирован вес a_i, показывающий относительную важность признака. Например, в задаче классификации документов признак f_i может соответствовать слову в документе, и с ним будет ассоциирован вес, равный оценке TF-IDF.

8.3. Соотношение между унитарным и плотным векторами

Представление признаков плотными векторами – неотъемлемая часть нейронных сетей, поэтому различия между использованием разреженного и плотного представлений признаков тоньше, чем может показаться на первый взгляд. На самом деле использование разреженных унитарных векторов в качестве входа при обучении нейронной сети сводится к тому, что первый слой сети обучается строить плотный вектор погружения для каждого признака на основе обучающих данных.

При использовании плотных векторов каждый категориальный признак f_i отображается в плотный d-мерный вектор $v(f_i)$. Это отображение выполняется с помощью *слоя погружения*, или слоя *поиска соответствия*. Рассмотрим словарь, содержащий $|V|$ слов, каждому из которых соответствует d-мерный вектор погружения. Этот набор векторов можно рассматривать как матрицу погружения $\boldsymbol{E}$ размера $|V| \times d$, каждая строка которой соответствует погруженному признаку. Обозначим $\boldsymbol{f_i}$ – унитарное представление признака f_i, т. е. $|V|$-мерный вектор, в котором равны нулю все элементы, кроме одного в i-й позиции – равного 1. Тогда произведение $\boldsymbol{f_i E}$ «выбирает» соответствующую строку $\boldsymbol{E}$. Следовательно, $v(f_i)$ можно определить в терминах $\boldsymbol{E}$ и $\boldsymbol{f_i}$:

$$v(f_i) = \boldsymbol{f_i E}. \tag{8.3}$$

И аналогично:

$$\text{CBOW}(f_1, \ldots, f_k) = \sum_{i=1}^{k} (\boldsymbol{f_i E}) = \left(\sum_{i=1}^{k} \boldsymbol{f_i}\right)\boldsymbol{E}. \tag{8.4}$$

Тогда вход сети рассматривается как набор унитарных векторов. Хотя это элегантная и математически корректная конструкция, для ее эффективной реализации обычно нужна основанная на хеш-таблице структура данных, которая отображает признаки на соответствующие им векторы погружения, не прибегая к унитарному представлению.

Рассмотрим сеть, в которой используется «традиционное» разреженное представление входных векторов без слоя погружения. В предположении, что множество всех признаков равно V и «включено» k признаков $f_1, \ldots, f_k$, $f_i \in V$, вход сети имеет вид:

$$\boldsymbol{x} = \sum_{i=1}^{k} \boldsymbol{f}_i, \quad \boldsymbol{x} \in \mathbb{N}_{+}^{|V|}, \tag{8.5}$$

и потому первый слой (опуская нелинейную функцию активации) выглядит так:

$$\boldsymbol{x}\boldsymbol{W} + \boldsymbol{b} = \left(\sum_{i=1}^{k} \boldsymbol{f}_i\right)\boldsymbol{W} + \boldsymbol{b}, \tag{8.6}$$
$$\boldsymbol{W} \in \mathbb{R}^{|V| \times d}, \boldsymbol{b} \in \mathbb{R}^{d}.$$

Этот слой выбирает строки $\boldsymbol{W}$, соответствующие входным признакам в $\boldsymbol{x}$, суммирует их и прибавляет смещение. Это очень похоже на слой погружения, который порождает представление признаков CBOW, в котором матрица $\boldsymbol{W}$ играет роль матрицы погружения. Основное отличие – вектор смещения $\boldsymbol{b}$ и тот факт, что слой погружения обычно не подвергается действию нелинейной функции активации, а передается напрямую первому слою. Другое отличие состоит в том, что в такой конфигурации каждый признак вынужденно получает отдельный вектор (строку $\boldsymbol{W}$), тогда как слой погружения предлагает больше гибкости, например разрешая признакам «следующее слово dog» и «предыдущее слово dog» разделять один и тот же вектор. Однако эти различия не слишком значительны. Когда дело доходит до многослойных сетей прямого распространения, выясняется, что различие между плотным и разреженным входами меньше, чем может показаться на первый взгляд.

8.4. Разные разности

8.4.1. Дистанционные и позиционные признаки

Линейное расстояние между двумя словами предложения может оказаться информативным признаком. Например, в задаче выделения событий[1] может быть задано ключевое слово и слово-кандидат на роль аргумента и требуется предсказать, действительно ли слово-кандидат является аргументом ключевого слова. Аналогично в задаче разрешения кореференции (решения о том, относится ли местоимение типа *he* или *she* к какой-нибудь из встретившихся ранее сущностей, и если да, то к какой именно) задается пара (местоимение, слово-кандидат) и требуется предсказать, кореферентны они или нет. Расстояние (или относительная позиция) между ключевым словом и аргументом – сильный сигнал для таких задач предсказания. В «традиционной» NLP расстояния обычно кодируются путем распределения по нескольким интервалам (например, 1, 2, 3, 4, 5–10, 10+) и ассо-

[1] Задача выделения событий заключается в идентификации событий из предопределенного списка типов. Например, речь может идти о событиях «покупка» или «террористическая атака». События каждого типа могут опознаваться по ключевым словам (обычно глаголам) и иметь несколько атрибутов (аргументов), которые необходимо определить (кто купил? что купил? в каком количестве?).

циирования с каждым интервалом унитарного вектора. В архитектуре нейронной сети, когда входной вектор не состоит из бинарных индикаторных признаков, может показаться естественным выделить по одному входному элементу для каждого дистанционного признака, присваивая ему в качестве значения расстояние. Однако на практике такой подход не применяется. Вместо этого дистанционные признаки кодируются так же, как признаки других типов: с каждым интервалом ассоциируется *d*-мерный вектор, и эти векторы погружения расстояния затем обучаются как обычные параметры сети [dos Santos et al., 2015, Nguyen and Grishman, 2015, Zeng et al., 2014, Zhu et al., 2015a].

8.4.2. Дополнение, неизвестные слова и прореживание слов

ДОПОЛНЕНИЕ Иногда экстрактор признаков ищет то, чего не существует. Например, при работе с деревьями разбора, возможно, имеется признак, который ищет самое левое слово, зависимое от данного, но так получилось, что слева зависимых слов нет. Или вас интересуют слова двумя позициями правее от текущего, но вы уже находитесь в конце предложения, и эта позиция оказывается за его пределами. Что делать в таких ситуациях? Если используется подход на основе мешка признаков (т. е. суммирование), то можно просто не включать признак в сумму. Если используется конкатенация, то можно подставить нулевой вектор. Технически эти два подхода работают прекрасно, но могут оказаться неоптимальными в конкретной предметной области. Что, если знание о том, что левый модификатор отсутствует, несет полезную информацию? Предлагаемое решение – добавить специальный символ (*символ дополнения*) в словарь погружения и использовать в таких случаях ассоциированный вектор дополнения. В зависимости от проблемы в разных ситуациях можно использовать разные векторы дополнения (т. е. отсутствию левого модификатора может соответствовать не тот вектор, что отсутствию правого). Такие дополнения важны для хорошего качества предсказания и часто применяются. К сожалению, об их использовании редко сообщается в научных статьях, а если и сообщается, то походя.

НЕИЗВЕСТНЫЕ СЛОВА Еще один случай, когда запрошенного вектора признаков нет в наличии, – *несловарные* слова. Вы ищете слово слева, видите значение, но обнаруживается, что это слово не входило в словарь на этапе обучения, поэтому для него нет вектора погружения. Этот случай отличается от дополнения, поскольку искомое слово присутствует, просто оно вам неизвестно. Однако решение аналогично – завести специальный символ Unk, представляющий неизвестную лексему. И снова могут понадобиться различные символы для разных словарей. В любом случае рекомендуется не использовать один и тот же вектор для дополнения и неизвестного слова, потому что эти два условия принципиально различны.

СИГНАТУРЫ СЛОВ Еще один метод борьбы с неизвестными словами – переход от словоформ к *сигнатурам слов*. Применение символа Unk для неизвестных слов, по существу, является переходом от всех неизвестных слов к единственной сигнатуре. Но в зависимости от решаемой задачи можно подобрать и более тонкие стратегии. Например, можно заменять неизвестные слова, оканчивающиеся на *ing*, символом *__ing*, слова, оканчивающиеся на *ed*, – символом *__ed*, слова, начинающиеся с *un*, – символом *un__*, числа – символом *NUM* и т. д. Список отображений составляется вручную и отражает информативные паттерны пере-

хода. Этот подход часто применяется на практике, но редко упоминается в статьях по глубокому обучению. Существуют методики, позволяющие автоматически обучаться такому поведению на этапе обучения модели, не прибегая к ручной работе (*см.* обсуждение субсловесных единиц в разделе 10.5.5), но во многих случаях это лишнее – вручную определять паттерны не сложнее и более эффективно с вычислительной точки зрения.

ПРОРЕЖИВАНИЕ СЛОВ Заведения специального вектора погружения для неизвестных слов недостаточно – если у всех признаков в обучающем наборе имеются собственные векторы погружения, то на этапе обучения неизвестных слов не встретится: ассоциированный вектор не будет обновляться, и модель не сможет настроиться на обработку неизвестных слов. Получается, что если неизвестное слово встречается на этапе тестирования, то мы просто берем случайный вектор. Модель должна видеть ситуацию с отсутствующими словами в процессе обучения. Одно из возможных решений – во время обучения заменить все (или некоторые) признаки с низкой частотой неизвестным символом (т. е. подвергнуть данные предварительной обработке, в ходе которой слова с частотой встречаемости ниже заданного порога заменяются словом *unknown*). Это решение работает, но у него есть недостаток – потеря части обучающих данных, поскольку редкие слова не получают никакого сигнала. Лучше использовать *прореживание слов* (word-dropout): при выделении признаков на этапе обучения случайным образом заменять слова неизвестным символом. Замена должна быть основана на частоте слова: редкие слова заменяются с большей вероятностью, чем частые. Решение о случайной замене следует принимать во время выполнения – слово, которое было заменено один раз, может быть удалено или не удалено, когда встретится заново (скажем, на разных проходах по набору данных). Не существует общепринятой формулы вычисления коэффициента прореживания. В нашей группе используется величина $\alpha/(\#(w)+\alpha)$, где α – параметр, управляющий агрессивностью прореживания [Kiperwasser and Goldberg, 2016b].

ПРОРЕЖИВАНИЕ СЛОВ КАК РЕГУЛЯРИЗАЦИЯ Помимо лучшей адаптации к неизвестным словам, прореживание может быть полезно для предотвращения переобучения и повышения надежности, поскольку не позволяет модели слишком сильно полагаться на присутствие какого-то одного слова [Iyyer et al., 2015]. При таком использовании прореживание следует часто применять не только к редким, но и к частым словам. На самом деле в работе Iyyer et al. [2015] предлагается применять к прореживанию слов испытания Бернулли с вероятностью p вне зависимости от их частоты. Если прореживание используется как регуляризатор, то не всегда имеет смысл заменять удаленные слова неизвестным символом. Например, если признаки представляются мешком слов в документе и удалено более четверти слов, то замена удаленных слов неизвестным символом создаст такое представление признаков, которое вряд ли встретится на этапе тестирования, когда столь высокая концентрация неизвестных слов маловероятна.

8.4.3. Комбинации признаков

Отметим, что на этапе выделения признаков в нейронной сети речь идет только о *базовых* признаках. Этим он отличается от традиционных систем NLP на основе линейной модели, когда конструктор признаков должен явно задавать не толь-

ко базовые признаки, но и взаимодействия между ними (например, вводить не только признак «слово X» и признак «метка части речи Y», но и комбинированный признак «слово X и метка части речи Y», а иногда даже ««слово X, метка части речи Y и предыдущее слово Z»). Комбинационные признаки очень важны в линейных моделях, поскольку увеличивают размерность входных данных, преобразуя их в пространство, где множество точек ближе к линейно разделимому. С другой стороны, пространство возможных комбинаций очень велико, и конструктор признаков должен тратить уйму времени на составление эффективного множества комбинаций. Одно из обещаний нелинейных моделей нейронных сетей заключается в том, что определять необходимо только базовые признаки. Ожидается, что благодаря нелинейности классификатор, определенный структурой сети, сможет сам найти характерные комбинации признаков, устраняя необходимость в ручном конструировании.

Как было сказано в разделе 3.3, ядерные методы [Shawe-Taylor and Cristianini, 2004], а особенно с полиномиальным ядром [Kudo and Matsumoto, 2003], также позволяют конструктору признаков задавать только базовые признаки, оставляя нахождение комбинаций алгоритму обучения. В отличие от нейросетевых моделей, ядерные методы выпуклые и, значит, допускают точное решение задачи оптимизации. Однако вычислительная сложность классификации в ядерных методах линейно зависит от размера обучающих данных, так что для практических целей они слишком медленные и не годятся для обучения на больших наборах данных. С другой стороны, вычислительная сложность классификации с использованием нейронных сетей линейно масштабируется с увеличением размера сети независимо от размера обучающего набора[1].

8.4.4. Обобществление векторов

Рассмотрим случай, когда имеется несколько признаков с общим словарем. Например, при назначении метки части речи данному слову может существовать набор признаков, учитывающий предыдущее слово, и набор признаков, учитывающий следующее слово. Строя входные данные для классификатора, мы конкатенируем векторные представления предыдущего и следующего слов. Тогда классификатор сможет различить два разных индикатора и обрабатывать их по-разному. Но должны ли оба признака разделять одни и те же векторы? Должен ли вектор для признака «dog:предыдущее-слово» быть таким же, как для признака «dog:следующее-слово»? Или нужно назначить им различные векторы? Этот вопрос также в основном эмпирический. Если вы полагаете, что слова, находящиеся в разных позициях, ведут себя по-разному (например, слово X ведет себя, как слово Y, если является предыдущим, но как слово Z, если следующим), то, возможно, имеет смысл использовать два разных словаря и сопоставить разные наборы векторов каждому типу признаков. Но если вы думаете, что слова ведут себя

[1] Конечно, во время обучения все равно необходимо обойти весь набор данных, а иногда и несколько раз. Поэтому время обучения линейно зависит от размера набора данных. Но каждый пример, будь то на этапе обучения или на этапе тестирования, обрабатывается за постоянное время (для данной сети). И в этом состоит отличие от ядерного классификатора, в котором время обработки каждого примера линейно растет с ростом набора данных.

одинаково в обеих позициях, то можно получить выигрыш, взяв общий словарь для обоих типов признаков.

8.4.5. Размерность

Сколько измерений следует выделить для каждого признака? К сожалению, по этому вопросу нет никаких теоретических границ и даже устоявшейся практики. Очевидно, что размерность должна расти с увеличением количества членов класса (вероятно, мы захотим отвести больше измерений погружениям слов, чем погружениям частей речи), но какой будет достаточно? В современных работах размерность векторов погружения слов варьируется от 50 до нескольких сотен, а в некоторых исключительных случаях доходит до нескольких тысяч. Поскольку от размерности векторов напрямую зависят требования к памяти и время обработки, рекомендуется поэкспериментировать с несколькими вариантами и выбрать приемлемый компромисс между быстродействием и верностью.

8.4.6. Словарь погружений

Что означает «ассоциировать вектор погружения для *каждого* слова»? Очевидно, что мы не можем ассоциировать вектор со всеми возможными значениями и должны ограничиться значениями из конечного словаря. Этот словарь обычно составляется на основе обучающего набора или, если используются предобученные погружения, на основе того набора, на котором эти погружения обучались. Рекомендуется включать в словарь также особый символ Unk, ассоциирующий специальный вектор со всеми словами, отсутствующими в словаре.

8.4.7. Выход сети

Для многоклассовых проблем классификации с k классами выходом сети является k-мерный вектор, в котором каждое измерение представляет силу одного выходного класса. То есть выход остается таким же, как в традиционных линейных моделях, – скалярные оценки объектов из дискретного множества. Однако, как мы видели в главе 4, с выходным слоем ассоциирована матрица размера $d \times k$, столбцы которой можно рассматривать как d-мерные погружения выходных классов. Сходство векторных представлений k классов отражает установленное моделью в ходе обучения сходство между выходными классами.

ИСТОРИЧЕСКОЕ ЗАМЕЧАНИЕ Представление слов плотными векторами для ввода в нейронную сеть популяризировано в работе Bengio et al. [2003] в контексте языкового моделирования нейронными сетями. На задачи, изучаемые в NLP, оно было распространено в пионерской работе Коллоберта, Уэстона и их сотрудников [Collobert and Weston, 2008, Collobert et al., 2011][1]. Применение погружений для представления не только слов, но и произвольных признаков было продемонстрировано широкой публике в работе Chen and Manning [2014].

[1] Хотя в работах Бенджио, Коллоберта, Уэстона и их коллег эти подходы были популяризированы, первыми их применили не они. Применение плотных векторов в непрерывном пространстве для представления слов, подаваемых на вход нейронным сетям, ранее изучалось в работах Lee et al. [1992] и Forcada and Ñeco [1997]. А для машинного перевода непрерывные языковые модели использовались в работе Schwenk et al. [2006].

8.5. Пример: частеречная разметка

В задаче о частеречной разметке (раздел 7.4) дана последовательность n слов w_1, w_2, ..., w_n и позиция слова i, а требуется предсказать метку w_i. В предположении, что слова помечаются слева направо, мы можем также получить доступ к предсказаниям предыдущих меток $\hat{p}_1$, ..., $\hat{p}_{i-1}$. Список конкретных базовых признаков приведен в разделе 7.4, здесь же мы обсудим, как закодировать их в виде входного вектора. Нам нужна функция выделения признаков $\boldsymbol{x} = \phi(s, i)$, которая получает последовательность s, содержащую слова, предыдущие метки и позицию i, а возвращает вектор признаков $\boldsymbol{x}$. Предположим, что имеется функция *suf*(w, k), возвращающая k-буквенный суффикс слова w, и функция *pref*(w, k), возвращающая префикс.

Начнем с трех булевых индикаторов: *word-is-capitalized* (слово начинается заглавной буквой), *word-contains-hyphen* (слово содержит дефис) и *word-contains-digit* (слово содержит цифру). Самый естественный способ закодировать их – ассоциировать с каждым отдельное измерение, которое будет равно 1, если соответствующее условие для слова w_i выполняется, и 0 в противном случае[1]. Поместим эти значения в трехмерный вектор $\boldsymbol{c}_i$, ассоциированный со словом i.

Далее мы должны закодировать слова, префиксы, суффиксы и метки частей речи в различных позициях окна. Ассоциируем с каждым словом w_i вектор погружения $\boldsymbol{v}_w(w_i) \in \mathbb{R}^{d_w}$. Аналогично ассоциируем с каждым двухбуквенным суффиксом *suf*(w_i, 2) вектор погружения $v_s(suf(w_i, 2))$, а с трехбуквенным суффиксом *suf*(w_i, 2) – вектор погружения $v_s(suf(w_i, 3))$, $v_s(\cdot) \in \mathbb{R}^{d_s}$. С префиксами поступим точно так же, определив погружения $v_p(\cdot) \in \mathbb{R}^{d_p}$. Наконец, каждой метке части речи сопоставляется погружение $\boldsymbol{v}_t(p_i) \in \mathbb{R}^{d_t}$. С каждой позицией i можно ассоциировать вектор $\boldsymbol{v}_i$ релевантной информации о слове (словоформа, префиксы, суффиксы, булевы признаки):

$$\boldsymbol{v}_i = [\boldsymbol{c}_i; v_w(w_i); v_s(suf(w_i, 2)); v_s(suf(w_i, 3)); v_p(pref(w_i, 2)); v_p(pref(w_i, 3))];$$
$$\boldsymbol{v}_i \in \mathbb{R}^{3+d_w+2d_s+2d_p}.$$

Тогда наш входной вектор $\boldsymbol{x}$ получается конкатенацией следующих векторов:

$$\boldsymbol{x} = \phi(s, i) = [\boldsymbol{v}_{i-2}; \boldsymbol{v}_{i-1}; \boldsymbol{v}_i; \boldsymbol{v}_{i+1}; \boldsymbol{v}_{i+2}; \boldsymbol{v}_t(p_{i-1}); \boldsymbol{v}_t(p_{i-2})];$$
$$\boldsymbol{v}_i \in \mathbb{R}^{5(3+d_w+2d_s+2d_p)+2d_t}.$$

ОБСУЖДЕНИЕ Заметим, что слова в разных позициях разделяют одни и те же векторы погружения – при создании $\boldsymbol{v}_i$ и $\boldsymbol{v}_{i-1}$ мы читаем одни и те же таблицы погружения – и что вектор $\boldsymbol{v}_i$ не «знает» свою относительную позицию. Однако благодаря конкатенации вектор $\boldsymbol{x}$ «знает», какова относительная позиция каждого включенного в него вектора $\boldsymbol{v}$, поскольку знает его позицию внутри $\boldsymbol{x}$. Это позволяет обобществить некоторую информацию между словами в разных позициях – вектор слова *dog* будет обновляться вне зависимости от того, находится это слово в относительной позиции –2 или +1, но при этом будет по-разному обрабатываться моделью, когда слово встречается в разных позициях, поскольку будет умножаться на разные части матрицы в первом слое сети.

[1] Можно также кодировать невыполнение условия значением –1.

Альтернативный подход – ассоциировать с каждой парой слово–позиция отдельное погружение, т. е. вместо одной таблицы v_w мы будем иметь пять таблиц погружения v_{w-2}, v_{w-1}, v_{w0}, v_{w+1}, v_{w+2} и использовать одну из них для каждой относительной позиции слова. При таком подходе существенно возрастает количество параметров модели (нам придется обучать в пять раз больше векторов погружения) и не достигается никакого обобществления между словами. Кроме того, это более расточительно в плане вычислений, потому что в первом подходе мы могли вычислять вектор v_i для каждого слова предложения только один раз, а затем использовать повторно при рассмотрении других позиций i, тогда как во втором векторы v_i придется заново вычислять для каждой позиции i. Наконец, будет труднее использовать предобученные векторы слов, поскольку с предобученными векторами не связана никакая информация о позиции. Однако второй подход позволил бы рассматривать позицию каждого слова абсолютно независимо от остальных, если это именно то, что нам нужно[1].

Еще один момент, который следует учитывать, – наличие заглавных букв. Должны ли погружения слов *Dog* и *dog* быть различными? Хотя заглавные буквы – важный ключ для частеречной разметки, в нашем случае информация о регистре первой буквы слова w_i уже закодирована в булевых признаках c_i. Поэтому рекомендуется привести все слова в словаре к нижнему регистру, перед тем как создавать или опрашивать таблицу погружений.

Наконец, признаки префикс-2 и префикс-3 взаимно избыточны (один содержит другой), и то же самое относится к суффиксам. Так ли нам нужны оба? Можно ли как-то разделить информацию между ними? На самом деле мы могли бы использовать погружения букв вместо погружения суффиксов и заменить оба погружения суффиксов вектором, равным конкатенации трех последних букв слова. В разделе 16.2.1 мы рассмотрим альтернативный подход, в котором используются рекуррентные нейронные сети (РНС) уровня литер, которые улавливают префикс, суффикс и разные другие свойства словоформы.

8.6. Пример: анализ методом разложения на дуги

В задаче анализа методом разложения на дуги (раздел 7.7) дана последовательность n слов $w_{1:n}$ и их меток частей речи $p_{1:n}$, а требуется предсказать дерево разбора. Здесь нас интересуют признаки для балльной оценки решения о связывании слов w_h и w_m, где w_h – кандидат на роль головного слова, а w_m – кандидат на роль слова-модификатора.

Список конкретных базовых признаков был приведен в разделе 7.7, а здесь мы обсудим, как закодировать их во входном векторе. Мы определим функцию выделения признаков $\boldsymbol{x} = \phi(h, m, sent)$, которая получает последовательность слов и меток частей речи, а также позиции заглавного слова (h) и слова-модификатора (m).

---

[1] Заметим, что, с точки зрения математики, это последнее преимущество и не преимущество вовсе – при использовании в качестве входа нейронной сети первый слой потенциально мог бы *обучиться* анализировать одни измерения погружений, когда они используются в позиции −1, и другие измерения, когда они используются в позиции +2. В результате было бы достигнуто такое же разделение, как в первом подходе. Следовательно, первый подход не менее выразителен, чем второй, по крайней мере теоретически.

Сначала необходимо рассмотреть заглавное слово, его метку, а также слова и метки в окне шириной 5 слов вокруг заглавного (по два слова с каждой стороны). Ассоциируем с каждым словом w нашего словаря вектор погружения $v_w(w) \in \mathbb{R}^{d_w}$, а с каждой меткой части речи p – вектор погружения $v_t(p) \in \mathbb{R}^{d_t}$. Затем определим векторное представление слова в позиции i как $\boldsymbol{v}_i = [v_w(w_i); v_t(p_i)] \in \mathbb{R}^{d_t+d_t}$, т. е. конкатенацию векторов слова и метки.

Далее ассоциируем с заглавным словом вектор $\boldsymbol{h}$, представляющий слово в контексте, а со словом-модификатором – аналогичный вектор $\boldsymbol{m}$:

$$\boldsymbol{h} = [\boldsymbol{v}_{h-2}; \boldsymbol{v}_{h-1}; \boldsymbol{v}_h; \boldsymbol{v}_{h+1}; \boldsymbol{v}_{h+2}];$$
$$\boldsymbol{m} = [\boldsymbol{v}_{m-2}; \boldsymbol{v}_{m-1}; \boldsymbol{v}_m; \boldsymbol{v}_{m+1}; \boldsymbol{v}_{m+2}].$$

Тем самым мы определили элементы первого блока признаков. Заметим, что, как и в случае признаков для частеречной разметки, в этом способе кодирования учтены относительные позиции каждого контекстного слова. Если бы эти позиции нас не интересовали, то можно было бы вместо этого представить заглавное слово вектором $\boldsymbol{h}' = [\boldsymbol{v}_h; (\boldsymbol{v}_{h-2}+\boldsymbol{v}_{h-1}+\boldsymbol{v}_{h+1}+\boldsymbol{v}_{h+2})]$. Здесь мы складываем контекстные слова в мешок слов, теряя позиционную информацию, но конкатенация контекстных слов с фокусным сохраняет различие между ними.

Обратимся теперь к *дистанционным* и *зависящим от направления* признакам. Можно было бы сопоставить расстоянию одно измерение, содержащее числовое значение расстояния, но обычно принято распределять расстояния по k дискретным интервалам (скажем, 1, 2, 3, 4–7, 8–10, 11+) и ассоциировать с каждым интервалом d_d-мерное погружение. Направление – это булев признак, мы представляем его отдельным измерением[1]. Обозначим $\boldsymbol{d}$ вектор, содержащий погружение этой гистограммы расстояний, конкатенированное с булевым направлением. Наконец, нам необходимо представить слова и метки частей речи между заглавным словом и модификатором. Их число не ограничено и изменяется в зависимости от примера, поэтому использовать конкатенацию нельзя. К счастью, нас не интересуют позиции промежуточных объектов, поэтому можно воспользоваться для кодирования мешком слов. Конкретно, будем представлять слова из промежуточного контекста вектором $\boldsymbol{c}$, определенным как среднее векторов промежуточных слов и меток:

$$\boldsymbol{c} = \sum_{i=h}^{m} \boldsymbol{v}_i.$$

Заметим, что эта сумма потенциально улавливает также количество элементов между заглавным словом и модификатором, так что дистанционный признак становится избыточным.

И наконец, в окончательном представлении решения о связывании, которое подлежит оценке, $\boldsymbol{x}$ кодируется как конкатенация различных элементов:

---

[1] Хотя эта кодировка измерения вполне естественна, авторы работы Pei et al. [2015] применили в своем анализаторе другой подход. Быть может, придавая особую важность информации о расстоянии, они решили не помечать его как входной признак, а использовать две разные функции оценивания: одну для дуг, направленных слева направо, а другую – для направленных справа налево. Это существенно повышает значимость информации о расстоянии, но вместе с тем увеличивает количество параметров модели.

$$\boldsymbol{x} = \phi(h, m, sent) = [\boldsymbol{h}; \boldsymbol{m}; \boldsymbol{c}; \boldsymbol{d}],$$

где

$$\begin{aligned}
\boldsymbol{v}_i &= [v_m(w_i); v_t(p_i)];\\
\boldsymbol{h} &= [\boldsymbol{v}_{h-2}; \boldsymbol{v}_{h-1}; \boldsymbol{v}_h; \boldsymbol{v}_{h+1}; \boldsymbol{v}_{h+2}];\\
\boldsymbol{m} &= [\boldsymbol{v}_{m-2}; \boldsymbol{v}_{m-1}; \boldsymbol{v}_m; \boldsymbol{v}_{m+1}; \boldsymbol{v}_{m+2}];\\
\boldsymbol{c} &= \sum_{i=h}^{m} \boldsymbol{v}_i;
\end{aligned}$$

$\boldsymbol{d}$ = погружения гистограмм расстояния; индикатор направления.

Обратите внимание, что мы комбинируем позиционные оконные признаки с признаками типа мешка слов с помощью простой конкатенации. Слои нейронной сети, расположенные выше $\boldsymbol{x}$, смогут вывести преобразование и комбинации признаков между элементами в разных окнах, а также между различными элементами мешка слов. Процесс создания представления $\boldsymbol{x}$ – таблицы погружений для слов, меток частей речи и гистограммы расстояний, а также различные операции конкатенации и суммирования – также является частью нейронной сети. Это отражено в построении графа вычислений, и параметры этого процесса обучаются вместе с сетью.

Та часть построения сети, которая связана с созданием признаков, может быть и еще сложнее. Например, если имеются основания полагать, что взаимодействия между словом и его меткой части речи, а также взаимодействия внутри контекстного окна более важны, чем взаимодействия между элементами разных сущностей, то это можно было бы отразить в кодировании входа, создав дополнительные нелинейные преобразования, т. е. заменив $\boldsymbol{v}_i$ вектором $\boldsymbol{v}'_I = g(\boldsymbol{v}_i\boldsymbol{W}^{\boldsymbol{v}} + \boldsymbol{b}^{\boldsymbol{v}})$, а $\boldsymbol{h}$ – вектором $\boldsymbol{h}' = g([\boldsymbol{v}'_{h-2}; \boldsymbol{v}'_{h-1}; \boldsymbol{v}'_h; \boldsymbol{v}'_{h+1}; \boldsymbol{v}'_{h+2}]\boldsymbol{W}^{\boldsymbol{h}} + \boldsymbol{b}^{\boldsymbol{h}})$ и положив $\boldsymbol{x} = [\boldsymbol{v}'; \boldsymbol{m}'; \boldsymbol{c}; \boldsymbol{d}]$.

Глава 9

Языковое моделирование

9.1. Задача языкового моделирования

Задача языкового моделирования заключается в том, чтобы назначить вероятность предложению, написанному на некотором языке (например, «какова вероятность встретить предложение *the lazy dog barked loudly*[1]?»). Помимо назначения вероятности каждой последовательности слов, языковая модель также назначает вероятность того, что данное слово (или последовательность слов) окажется после заданной последовательности слов («какова вероятность встретить слово *barked* после фразы *the lazy dog*?»)[2].

Считается, что качество решения задачи языкового моделирования идеально, если количество попыток предсказания следующего слова последовательности машиной меньше или равно количеству попыток, которое понадобилось бы человеку. Это признак интеллекта человеческого уровня[3], но маловероятно, что такое качество будет достигнуто в ближайшем будущем. Но даже при неидеальном качестве языковое моделирование является важнейшим компонентом практических приложений, например машинного перевода и автоматического распознавания речи, когда система порождает несколько гипотез о переводе или

---

[1] Ленивая собака громко лаяла. – *Прим. перев.*

[2] Заметим, что умение назначать вероятность следования слова за последовательностью слов $p(w_{i_1}|w_1, w_2, \ldots, w_{i-1})$ и умение назначать вероятность произвольной последовательности слов $p(w_1, w_2, \ldots, w_k)$ эквивалентны, поскольку одна выводится из другой. Предположим, что мы умеем моделировать вероятности последовательностей. Тогда условную вероятность слова можно записать в виде отношения вероятностей двух последовательностей:

$$p(w_i|w_1, w_2, \ldots, w_{i-1}) = \frac{p(w_1, w_2, \ldots, w_{i-1}, w_i)}{p(w_1, w_2, \ldots, w_{i-1})}.$$

Наоборот, если бы мы умели моделировать условную вероятность следования слова за последовательностью слов, то могли бы воспользоваться цепным правилом, чтобы выразить вероятность последовательности как произведение условных вероятностей:

$p(w_1, \ldots, w_k) = p(w_1|<s>)\times p(w_2|<s>, w_1)\times p(w_3|<s>, w_1, w_2)\times\ldots\times p(w_k|<s>, w_1, \ldots, w_{k-1})$,

где $<s>$ – специальный символ, обозначающий начало последовательности.

[3] Действительно, любой вопрос можно сформулировать как задачу угадывания следующего слова, например: «*Ответом на вопрос X является* ___». Но даже без таких патологических примеров предсказание следующего слова в тексте требует понимания синтаксических и семантических правил языка, а также значительного объема знаний о мире.

транскрипции, которые затем оцениваются языковой моделью. Поэтому языковое моделирование играет центральную роль в обработке естественного языка, искусственном интеллекте и исследованиях по машинному обучению.

Формально задача языкового моделирования ставится так: назначить вероятность произвольной последовательности слов $w_{1:n}$, т. е. оценить величину $P(w_{1:n})$. Применив цепное правило исчисления вероятностей, мы сможем записать эту вероятность в виде:

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2})P(w_4|w_{1:3}) \ldots P(w_n|w_{1:n-1}), \tag{9.1}$$

т. е. в виде последовательности задач предсказания слова, в которой каждое предсказание обусловлено предшествующими словами. Хотя задача моделирования одного слова на основе его левого контекста кажется проще, чем назначение оценки вероятности всему предложению, последний член в этом равенстве все равно требует обусловливания $n - 1$ словами, а это такая же трудная задача, как моделирование всего предложения. Поэтому в языковых моделях применяется *марковское предположение*, утверждающее, что будущее не зависит от прошлого при условии настоящего. Формально марковское предположение k-го порядка означает, что следующее слово в последовательности зависит только от k последних слов:

$$P(w_{i+1}|w_{1:i}) \approx P(w_{i+1}|w_{i-k:i}).$$

Тогда оценка вероятности предложения принимает вид:

$$P(w_{1:n}) \approx \prod_{i=1}^{n} P(w_i|w_{i-k:i-1}), \tag{9.2}$$

где $w_{-k+1}, \ldots, w_0$ – специальные символы дополнения.

Тогда наша задача состоит в том, чтобы дать верную оценку $P(w_{i+1}|w_{i-k:i})$ при наличии больших объемов текста.

Хотя марковское предположение k-го порядка, очевидно, неверно для всех k (в предложении могут встречаться произвольно длинные зависимости; в качестве простого примера рассмотрим сильную зависимость между первым словом предложения *what* (какой) и последним *?*), оно все же дает хорошие результаты для относительно малых k и являлось преобладающим подходом к языковому моделированию на протяжении нескольких десятков лет. В этой главе обсуждаются языковые модели k-го порядка, а в главе 14 мы разберем методы языкового моделирования без применения марковского предположения.

9.2. Оценивание языковых моделей: перплексивность

Существует несколько метрик для оценки языковых моделей. Системные метрики вычисляются в контексте решения задачи более высокого уровня, например путем измерения улучшения качества перевода при замене компоненты языкового моделирования A компонентой B в системе машинного перевода.

Внутренней оценкой языковых моделей является *перплексивность* для непредъявлявшихся предложений. Перплексивность – это теоретико-информационная мера, показывающая, насколько хорошо вероятностная модель предсказывает выборку. Чем меньше перплексивность, тем лучше аппроксимация. Если дан корпус текстов, содержащий n слов $w_1, \ldots, w_n$ (n может исчисляться миллионами), и функция языковой модели *LM*, назначающая вероятность слову на основе его истории, то перплексивность *LM* относительно корпуса равна:

$$2^{-\frac{1}{n}\sum_{i=1}^{n}\log_2 \mathrm{LM}(w_i|w_{1:i-1})}.$$

Хорошие языковые модели (отражающие реальное использование языка) назначают высокие вероятности событиям из корпуса, а стало быть, характеризуются низкой перплексивностью.

Перплексивность – хороший индикатор качества языковой модели[1]. Перплексивность зависит от корпуса – перплексивность двух языковых моделей можно сравнивать только относительно одного и того же корпуса.

9.3. Традиционные подходы к языковому моделированию

При традиционном подходе к языковой модели предполагается выполнение марковского свойства k-го порядка, так что справедливо соотношение $P(w_{i+1} = m \mid w_{1:i}) \approx P(w_{i+1} = m \mid w_{i-k:i})$. Роль языковой модели в этом случае сводится к вычислению хороших оценок $\hat{p}(w_{i+1} = m \mid w_{i-k:i})$. Оценки обычно выводятся на основе счетчиков слов в корпусе. Обозначим $\#(w_{i:j})$ счетчик последовательности слов $w_{i:j}$ в корпусе. Тогда оценка максимального правдоподобия (ОМП) $\hat{p}(w_{i+1} = m \mid w_{i-k:i})$ равна:

$$\hat{p}_{\mathrm{MLE}}(w_{i+1} = m | w_{i-k:i}) = \frac{\#(w_{i-k:i+1})}{\#(w_{i-k:i})}.$$

Хотя этот подход эффективен, у него есть один крупный недостаток: если событие $w_{i-k:i+1}$ никогда не встречалось в корпусе ($\#(w_{i-k:i+1}) = 0$), то ему будет назначена вероятность 0, а это, в свою очередь, приведет к назначению нулевой вероятности всему корпусу в силу мультипликативной природы вычисления вероятности предложения. Нулевая вероятность означает бесконечно большую перплексивность, что очень плохо. События с нулевой вероятностью – дело обычное: рассмотрим триграммную языковую модель, которая обусловливается только двумя словами, и словарь на 10 000 слов (довольно маленький). Существует $10\,000^3 = 10^{12}$ возможных троек слов, и понятно, что многие из них не будут встречаться в обучающем корпусе, содержащем, скажем, 10^{10} слов. Большинство из этих со-

[1] Однако важно отметить, что во многих случаях улучшение перплексивности не приводит к улучшению внешних оценок качества решения задачи. В этом смысле перплексивность хороша для сравнения языковых моделей с точки зрения их способности находить регулярность в последовательностях, но не очень хороша для оценки прогресса в понимании языка или решения задач лингвистической обработки.

бытий не встречается, потому что они не имеют смысла, но есть и много вполне осмысленных, но отсутствующих в корпусе.

Один из способов избежать событий с нулевой вероятностью – использование *методов сглаживания*, которые гарантируют, что какая-то (возможно, очень небольшая) масса вероятности будет назначена каждому возможному событию. Простейший пример – аддитивное сглаживание, называемое также *сглаживанием add-α* [Chen and Goodman, 1999, Goodman, 2001, Lidstone, 1920]. В нем предполагается, что каждое событие произошло по крайней мере α раз, помимо тех, что наблюдаются в корпусе. Тогда оценка принимает вид:

$$\hat{p}_{\text{add-}\alpha}(w_{i+1} = m | w_{i-k:i}) = \frac{\#(w_{i-k:i+1}) + \alpha}{\#(w_{i-k:i}) + \alpha |V|},$$

где $|V|$ – размер словаря и $0 < \alpha \leq 1$. Существуют и более изощренные методы сглаживания.

Другое популярное семейство решений основано на использовании *отката* (back-off): если k-грамма не наблюдалась, вычислить оценку на основе $(k-1)$-граммы. Репрезентативный пример из этого семейства дает *интерполированное сглаживание Джелинека–Мерсера* [Chen and Goodman, 1999, Jelinek and Mercer, 1980]:

$$\hat{p}_{\text{int}}(w_{i+1} = m | w_{i-k:i}) = \lambda_{w_{i-k:i}} \frac{\#(w_{i-k:i+1})}{\#(w_{i-k:i})} + (1 - \lambda_{w_{i-k:i}}) \hat{p}_{\text{int}}(w_{i+1} = m | w_{i-(k-1):i}).$$

Для достижения оптимального качества значения $\lambda_{w_{i-k:i}}$ должны зависеть от содержания обусловливающего контекста $w_{i-k:i}$: редкие контексты должны обрабатываться иначе, нежели частые.

В современных методах языкового моделирования, не основанных на нейронных сетях, используется *модифицированное сглаживание Кнезера–Нея*, являющееся вариантом метода, предложенного в работе Kneser and Ney [1995]. Детали *см.* в работах Chen and Goodman [1996] и Goodman [2001].

9.3.1. Для дальнейшего чтения

Языковое моделирование – очень обширная тема, разрабатываемая уже несколько десятилетий. Хороший формальный обзор, а также обоснование перплексивности как меры качества можно найти в материалах к лекциям Майкла Коллинза[1]. Обзор и эмпирические результаты по методам сглаживания имеются также в работах Chen and Goodman [1999] и Goodman [2001]. Еще один обзор традиционных методов языкового моделирования *см.* во вводных главах докторской диссертации Mikolov [2012]. Последние достижения в области языкового моделирования без применений нейронных сетей описаны в работе Pelemans et al. [2016].

9.3.2. Ограничения традиционных языковых моделей

Подходы к языковому моделированию на основе сглаженных оценок ОМП («традиционные») допускают простое обучение, масштабируются на большие корпусы

[1] http://www.cs.columbia.edu/~mcollins/lm-spring2013.pdf.

текстов и хорошо работают на практике. Однако у них есть несколько серьезных недостатков.

Методы сглаживания сложны для понимания и основаны на откате к событиям более низкого порядка. Это предполагает фиксированный порядок отката, который приходится проектировать вручную, что затрудняет добавление более «творческих» контекстов обусловливания (т. е. если мы захотим обусловливать k предшествующими словами и жанром текста, то как должен поступить алгоритм отката: сначала отбросить k-е предшествующее слово или жанр?). Последовательная природа отката усложняет масштабирование на более широкие n-граммы, чтобы уловить дальние зависимости: чтобы уловить зависимость между следующим словом и предшествующим ему словом, отстоящим на 10 позиций, необходимо видеть в тексте релевантную 11-грамму. На практике очень редко случается, чтобы модель откатывалась далеко назад. Быть может, было бы лучше пропускать при откате промежуточные слова, т. е. допустить n-граммы с «дырками». Однако это трудно сделать, не жертвуя правильностью порождающего вероятностного каркаса[1].

Масштабирование на более длинные n-граммы – проблема, внутренне присущая языковым моделям на основе ОМП. В силу самой природы естественного языка и огромного количества слов в словаре статистика для длинных n-грамм по необходимости будет разреженной. Кроме того, масштабирование на длинные n-граммы очень дорого обходится с точки зрения требований к памяти. Количество возможных n-грамм для словаря размера V равно $|V|^n$: увеличение ширины на 1 приводит к увеличению этого числа в $|V|$ раз. И хотя не все теоретически возможные n-граммы допустимы или встречаются в тексте, количество наблюдаемых событий кратно возрастает при увеличении длины n-граммы на 1. Поэтому обрабатывать контексты с более длинным обусловливанием оказывается очень накладно.

Наконец, языковые модели на основе ОМП страдают от недостатка обобщаемости на новые контексты. Даже если мы видели словосочетания *black car* и *blue car*, это никак не повлияет на оценку вероятности события *red car*, если мы не встречали его прежде[2].

9.4. Нейросетевые языковые модели

Нелинейные нейросетевые модели устраняют некоторые недостатки традиционных языковых моделей: они допускают обусловливание контекстами возрастающей длины ценой лишь линейного увеличения количества параметров, позволяют отказаться от ручного проектирования порядка отката и поддерживают обобщение на новые контексты.

---

[1] Впрочем, смотрите исследования по факторным языковым моделям (например, A. Bilmes and Kirchhoff [2003]) и (логлинейным) языковым моделям с максимальной энтропией, начатые работой A. Bilmes and Kirchhoff [2003], а также недавнюю работу в этом направлении Pelemans et al. [2016].

[2] В языковых моделях на основе классов [Brown et al., 1992] эту проблему пытаются решить путем кластеризации слов с помощью дистрибутивных алгоритмов и обусловливать индуцированными классами слов вместо самих слов или вместе с ними.

Вид модели, представленный в этой главе, популяризирован в работе Bengio et al. [2003].[1]

Входом нейронной сети является k-грамма, состоящая из слов $w_{1:k}$, а выходом – распределение вероятностей следующего слова. k контекстных слов $w_{1:k}$ рассматриваются как окно слова: с каждым словом w ассоциируется вектор погружения $v(w) \in \mathbb{R}^{d_w}$, и входной вектор $\boldsymbol{x}$ является конкатенацией k слов:

$$\boldsymbol{x} = [v(w_1); v(w_1); \ldots v(w_k)].$$

Затем вход $\boldsymbol{x}$ загружается в МСП с одним или несколькими скрытыми слоями:

$$\begin{aligned} \hat{\boldsymbol{y}} = P(w_i | w_{1:k}) = LM(w_{1:k}) &= \text{softmax}(\boldsymbol{h}\boldsymbol{W}^2 + \boldsymbol{b}^2); \\ \boldsymbol{h} &= g(\boldsymbol{x}\boldsymbol{W}^1 + \boldsymbol{b}^1); \\ \boldsymbol{x} &= [v(w_1); v(w_2); \ldots; v(w_k)]; \\ v(w) &= \boldsymbol{E}[w]; \end{aligned} \tag{9.3}$$

$$w_i \in V; \quad \boldsymbol{E} \in \mathbb{R}^{|V| \times d_w}; \quad \boldsymbol{W}^1 \in \mathbb{R}^{k \cdot d_w \times d_{\text{hid}}}; \quad \boldsymbol{b}^1 \in \mathbb{R}^{d_{\text{hid}}}; \quad \boldsymbol{W}^2 \in \mathbb{R}^{d_{\text{hid}} \times |V|}; \quad \boldsymbol{b}^2 \in \mathbb{R}^{|V|}.$$

Здесь V – конечный словарь, включающий уникальные символы Unk для неизвестных слов, <s> – для дополнения предложения в начале и </s> – для обозначения конца предложения. Размер словаря $|V|$ варьируется от 10 000 до 1 000 000 слов, причем наиболее распространен размер примерно 70 000.

ОБУЧЕНИЕ Обучающими примерами являются просто словесные k-граммы, взятые из корпуса текстов, причем идентификаторы первых $k - 1$ слов используются в качестве признаков, а последнее слово – как целевая метка для классификации. Концептуально для обучения модели применяется перекрестная энтропия в качестве функции потери. Такая потеря дает очень хорошие результаты, но необходимо использовать дорогостоящую функцию softmax, из-за которой приходится отказываться от очень больших словарей. Поэтому представляет интерес поиск альтернативных функций потери или аппроксимаций softmax (*см.* ниже).

ТРЕБОВАНИЯ К ПАМЯТИ И ВЫЧИСЛИТЕЛЬНАЯ ЭФФЕКТИВНОСТЬ Каждое из k входных слов вносит d_w измерений в $\boldsymbol{x}$, а переход от k к $k + 1$ увеличивает размер матрицы весов $\boldsymbol{W}^1$ с $k \cdot d_w \cdot d_{\text{hid}}$ до $(k + 1) \cdot d_w \cdot d_{\text{hid}}$ – небольшое линейное увеличение числа параметров в противоположность полиномиальному росту в случае традиционных, основанных на счетчиках моделей. Это возможно, потому что комбинации признаков вычисляются в скрытом слое. Увеличение k, скорее всего, потребует также увеличения размера d_{hid}, но это все равно очень скромный рост числа параметров по сравнению с традиционными моделями. Добавление нелинейных слоев для улавливания более сложных взаимодействий также обходится относительно дешево.

С каждым словарным словом ассоциирован один d_w-мерный вектор (строка $\boldsymbol{E}$) и один d_{hid}-мерный вектор (столбец $\boldsymbol{W}^2$). Таким образом, добавление слов приводит к линейному увеличению количества параметров, что опять-таки много лучше, чем в традиционном случае. Однако, хотя во входном словаре (матрица $\boldsymbol{E}$)

[1] Похожая модель была описана еще в 1988 году в работе Nakamura and Shikano [1988] по предсказанию классов слов с помощью нейронных сетей.

производится только поиск и его рост не оказывает существенного влияния на скорость вычислений, размер выходного словаря такое влияние оказывает: применение функции softmax к выходному слою подразумевает дорогостоящее умножение вектора на матрицу $\boldsymbol{W}^2 \in \mathbb{R}^{d_{\text{hid}} \times |V|}$, за которым следует $|V|$ операций возведения в степень. Это вычисление определяет время выполнения, поэтому языковое моделирование с большим размером словаря оказывается невозможным.

БОЛЬШИЕ ВЫХОДНЫЕ ПРОСТРАНСТВА Препятствием для применения нейросетевых вероятностных языковых моделей с большим выходным пространством (т. е. для вычисления softmax по большому словарю) может оказаться неприемлемо высокое время обучения и тестирования. Поиск эффективных методов работы с большими выходными пространствами – тема активных исследований. Ниже описаны некоторые из существующих решений.

Иерархическая функция softmax [Morin and Bengio, 2005] позволяет вычислять вероятность одного слова за время $O(\log |V|)$, а не $O(|V|)$. Это достигается путем организации вычисления softmax в виде обхода дерева, когда вероятность каждого слова равна произведению решений о выборе ветви. В предположении, что нас интересует вероятность одного слова (а не распределение вероятностей всех слов), такой подход имеет очевидные преимущества в плане времени обучения и тестирования.

Решения на основе самонормировки, например *шумосопоставительное оценивание* (noise-contrastive estimation – NCE) [Mnih and Teh, 2012, Vaswani et al., 2013] или добавление нормировочного члена к целевой функции обучения [Devlin et al., 2014]. Подход NCE улучшает время обучения благодаря замене перекрестной энтропии набором задач бинарной классификации, которые требуют вычисления оценок для k случайно выбранных слов, а не для всего словаря. Этот метод позволяет также улучшить предсказание на этапе тестирования, подталкивая модель в сторону порождения «приблизительно нормированных» экспоненциальных оценок, что делает выданную моделью оценку слова хорошей заменой его вероятности. Подход на основе нормировочного члена (Devlin et al. [2014]) также повышает эффективность на этапе тестирования за счет добавления в целевую функцию обучения члена, который поощряет выбор моделью таких экспоненциальных оценок, которые в сумме дают единицу, что избавляет от необходимости явного суммирования в процессе тестирования (этот метод не повышает эффективность на этапе обучения).

Выборочные методы аппроксимируют вычисление softmax на этапе обучения на относительно небольшом подмножестве словаря [Jean et al., 2015].

Хороший обзор и сравнение этих и других методов работы с большими выходными словарями приведен в статье Chen et al. [2016].

Принципиально другим направлением является попытка обойти проблему посредством работы на уровне литер, а не слов.

ЖЕЛАТЕЛЬНЫЕ СВОЙСТВА Если оставить в стороне запретительную стоимость использования большого выходного словаря, то у модели имеется ряд весьма привлекательных свойств. Она позволяет достичь лучшей перплексивности, чем современные традиционные языковые модели, например со сглаживанием Кнезера–Нея, и может масштабироваться на гораздо большие порядки, чем традиционные модели. Это возможно, потому что параметры ассоциируются с от-

дельными словами, а не с *k*-граммами. Также слова в разных позициях разделяют общие параметры, а значит, и их статистическую силу. Скрытые слои модели отвечают за нахождение информативных комбинаций слов и могут, по крайней мере теоретически, выявить тот факт, что для некоторых слов информативны только части *k*-граммы: модель может обучиться при необходимости откатываться к меньшим *k*-граммам, как в традиционной языковой модели, и делать это контекстно-зависимым способом. Она также может обучиться skip-граммам, т. е. для некоторых комбинаций рассматривать только слова в позициях 1, 2 и 5, пропуская слова 3 и 4[1].

Еще одно привлекательное свойство модели, помимо гибкости в отношении порядка *k*-грамм, – обобщаемость на новые контексты. Например, наблюдая слова *blue*, *green*, *red*, *black* и т. п. в похожих контекстах, модель может назначить разумную оценку событию *green car*, хотя никогда не видела этой комбинации в процессе обучения, – просто потому, что видела комбинации *blue car* и *red car*.

Сочетание этих свойств – гибкости при рассмотрении только частей обусловливающего контекста и способности к обобщению на ранее не виденные контексты – вкупе с линейной зависимостью объема потребной памяти и скорости вычислений от размера обусловливающего контекста позволяет без труда увеличивать размер обусловливающего контекста, не опасаясь резкого роста разреженности данных и времени вычислений.

Легкость, с которой нейросетевая языковая модель может включать в себя большие и гибкие обусловливающие контексты, позволяет творчески подходить к определению контекста. Например, в работе Devlin et al. [2014] предложена модель машинного перевода, в которой вероятность следующего слова обусловлена предшествующими *k* словами сгенерированного перевода, а также *m* словами на исходном языке, на которых основана заданная позиция в переводе. В результате модель оказывается чувствительной к тематической терминологии и состоящим из нескольких слов выражениям на исходном языке и действительно дает гораздо лучшие оценки перевода.

ОГРАНИЧЕНИЯ Представленные выше нейросетевые модели имеют некоторые ограничения: предсказание вероятности слова в контексте гораздо дороже, чем при использовании традиционных моделей, а работа с большими словарями и корпусами обучающих текстов может оказаться невозможной из-за непомерных затрат. Однако они лучше используют доступные данные и могут давать вполне конкурентоспособную перплексивность даже при относительно небольшом размере обучающего набора.

В системах машинного перевода применение нейросетевых языковых моделей не всегда улучшает качество перевода по сравнению с традиционными языковыми моделями со сглаживанием Кнезера–Нея. Однако качество перевода все же улучшается, когда вероятности, вычисленные традиционной и нейросетевой моделями, интерполируются. Похоже, что обе модели дополняют друг друга: нейросетевая модель лучше обобщается на ранее не встречавшиеся события, но иногда обобщаемость вредит качеству и предпочтительнее оказывается жесткость тра-

[1] Такие skip-граммы изучались и для языковых моделей без применения нейронных сетей; *см*. работу Pelemans et al. [2016] и приведенные в ней ссылки.

дицинных моделей. В качестве примера рассмотрим задачу, противоположную вышеупомянутому примеру с цветами: требуется назначить вероятность предложению *red horse* (красная лошадь). Традиционная модель назначит ему очень низкую оценку, поскольку такое событие встречалось всего несколько раз, если вообще встречалось. С другой стороны, нейросетевая модель, возможно, видела словосочетания *brown horse*, *black horse* и *white horse* и независимо обучилась тому, что слова *black*, *white*, *brown* и *red* могут встречаться в похожих контекстах. Такая модель назначит событию *red horse* куда более высокую вероятность, что нежелательно.

9.5. Использование языковых моделей для порождения

Языковые модели можно использовать также для порождения предложений. После обучения модели на заданном наборе текстов мы можем сгенерировать выборку случайных предложений из модели в соответствии с обученным ей распределением вероятностей. Для этого применяется следующий процесс: предсказать распределение вероятностей первого слова при условии начального символа и выбрать случайное слово в соответствии с этим распределением. Затем предсказать распределение вероятностей второго слова при условии первого и т. д., пока не будет предсказан символ конца последовательности </s>. Уже при $k = 3$ получается вполне приемлемый текст, а с увеличением k его качество растет.

При таком декодировании обученной языковой модели (генерировании предложения) на каждом шаге можно выбирать либо предсказание (слово) с наивысшей оценкой, либо случайное слово из предсказанного распределения. Еще один вариант – использовать *лучевой поиск* для нахождения предложения с глобально высокой вероятностью (если брать на каждом шаге предсказание с наивысшей вероятностью, то в результате иногда получается неоптимальная полная вероятность, потому что процесс может «загнать себя в угол», выбрав префикс, за которым следуют только события с низкой вероятностью). Эта ситуация, называемая проблемой *смещения метки*, подробно обсуждается в работах Andor et al. [2016] и Lafferty et al. [2001].

ВЫБОРКА ИЗ МУЛЬТИНОМИАЛЬНОГО РАСПРЕДЕЛЕНИЯ Мультиномиальное распределение $|V|$ элементов сопоставляет каждому элементу $0 < i \leq |V|$ вероятность $p_i \geq 0$ таким образом, что $\sum_{i=1}^{|V|} p_i = 1$. Для выборки случайного элемента из мультиномиального распределения можно применить следующий алгоритм:

$i \leftarrow 0$
$s \sim U[0, 1]$ ▷ случайное число, равномерно распределенное между 0 и 1
while $s \geq 0$ **do**
    $i \leftarrow i + 1$
    $s \leftarrow s - p_i$
return i

Это наивный алгоритм, вычислительная сложность которого линейно зависит от размера словаря – $O(|V|)$. Для больших словарей он может оказаться недопустимо медленным. Для островершинных распределений, когда значения отсортированы в порядке убывания вероятностей, среднее время будет значительно меньше. *Метод Уолкера* (alias method) [Kronmal and Peterson, Jr., 1979] – эффективный алгоритм для выборки из произвольных мультиномиальных распределений с большим словарем, который работает за время $O(1)$ после предобработки, занимающей линейное время.

9.6. Побочный продукт: представления слов

Языковые модели можно обучать на *необработанном тексте*: для обучения модели k-го порядка нужно только выделить $(k+1)$-граммы из текста и рассматривать $(k+1)$-е слово как сигнал учителя. Таким образом мы можем сгенерировать для них практически неограниченный объем обучающих данных.

Рассмотрим матрицу $\boldsymbol{W}^2$, существующую непосредственно перед окончательным применением функции softmax. Каждый столбец этой матрицы – d_{hid}-мерный вектор, ассоциированный с элементом словаря. В процессе вычисления финальной оценки каждый столбец $\boldsymbol{W}^2$ умножается на представление контекста $\boldsymbol{h}$, в результате чего порождается оценка соответствующего элемента словаря. Интуитивно очевидно, что при этом словам, с большой вероятностью встречающимся в похожих контекстах, должны соответствовать похожие векторы. Согласно дистрибутивной гипотезе, слова, встречающиеся в похожих контекстах, имеют похожий смысл, а значит, словам с похожим смыслом соответствуют похожие векторы. Аналогичное рассуждение применимо к строкам матрицы погружения $\boldsymbol{E}$. В качестве побочного продукта процесса языкового моделирования мы заодно обучаем полезное представление слов строками и столбцами матриц $\boldsymbol{E}$ и $\boldsymbol{W}^2$.

В следующей главе мы продолжим изучение вопроса о том, как обучать полезные представления слов, имея только необработанный текст.

Глава 10

Предобученные представления слов

Основной компонент нейросетевого подхода – использование погружений, т. е. представление каждого признака вектором в пространстве низкой размерности. Но откуда берутся эти векторы? В настоящей главе описываются общепринятые подходы.

10.1. Случайная инициализация

Если данных для обучения с учителем достаточно, то можно рассматривать погружения признаков так же, как прочие параметры модели: инициализировать векторы погружения случайными значениями и позволить процедуре обучения сети сделать из них «хорошие» векторы.

К случайной инициализации следует подходить аккуратно. В эффективной реализации алгоритма Word2Vec [Mikolov et al., 2013b,a] векторы слов инициализируются случайными числами, равномерно распределенными в диапазоне $\left[-\frac{1}{2d}, \frac{1}{2d}\right]$, где d – число измерений. Другой вариант – воспользоваться *инициализацией Ксавье* (*см.* раздел 5.2.2), когда выбираются значения, равномерно распределенные в диапазоне $\left[-\frac{\sqrt{6}}{\sqrt{d}}, \frac{\sqrt{6}}{\sqrt{d}}\right]$.

На практике рандомизация часто используется для инициализации векторов погружения таких распространенных признаков, как метки частей речи или отдельные буквы, а для инициализации потенциально редких признаков, например отдельных слов, применяется какая-то форма предобучения с учителем или без учителя. Далее предобученные векторы можно трактовать либо как фиксированные на весь процесс обучения сети, либо – и так поступают чаще – как случайно инициализированные векторы, которые затем настраиваются под решаемую задачу.

10.2. Специализированное предобучение с учителем

Если нас интересует задача A, для которой количество размеченных данных ограничено (например, задача синтаксического разбора), но имеется вспомогатель-

ная задача B (скажем, частеречная разметка), для которой размеченных данных достаточно, то можно предварительно обучить векторы слов, которые будут хорошими предикторами для задачи B, а затем использовать уже обученные векторы для обучения задачи A. Таким образом мы сможем задействовать большой объем размеченных данных, имеющихся для задачи B. В процессе обучения задачи A предобученные векторы можно либо считать фиксированными, либо произвести их дополнительную настройку специально для задачи A. Еще один вариант – обучать совместно для обеих целей (подробнее *см*. главу 20).

10.3. Предобучение без учителя

Часто встречается ситуация, когда не существует вспомогательной задачи с большим объемом аннотированных данных (или, быть может, мы хотим приступить к обучению вспомогательной задачи с более качественными начальными векторами). В таких случаях мы прибегаем к обучению вспомогательных задач «без учителя» на огромных корпусах неаннотированного текста.

Методы обучения векторов слов, по существу, такие же, как при обучении с учителем, но вместо обучения интересующей нас задачи мы создаем практически неограниченное количество обучающих примеров из необработанного текста, надеясь, что обученные таким образом задачи будут совпадать (или окажутся достаточно близки) с той, что мы в действительности решаем[1].

Основная идея методов, не требующих учителя, заключается в том, чтобы векторы погружения «похожих» слов были похожи. Хотя дать определение схожести слов трудно и обычно оно зависит от задачи, все современные подходы основаны на дистрибутивной гипотезе [Harris, 1954], утверждающей, что *слова похожи, если они встречаются в похожих контекстах*. Все имеющиеся методы, как бы различны они ни были, создают примеры для обучения с учителем, цель которых – предсказать слово по контексту или контекст по слову.

В последнем разделе главы 9 мы видели, что побочным продуктом языкового моделирования является создание векторов слов. На самом деле языковое моделирование можно рассматривать как обучение без учителя, при котором предсказание слова основано на контексте, состоящем из k предшествующих слов. Исторически алгоритм Коллоберта и Уэстона [Collobert and Weston, 2008, Collobert et al., 2011] и описанное ниже семейство алгоритмов Word2Vec [Mikolov et al., 2013b,a] основывались именно на этом свойстве языкового моделирования. Алгоритмы Word2Vec проектировались с учетом достижений тех же побочных эффектов, что в языковом моделировании, но с помощью более эффективного и гибкого каркаса. Алгоритм GloVe, описанный в работе Pennington et al. [2014], преследует ту же цель. Эти алгоритмы имеют глубокие связи с другим семейством алгоритмов, которые разрабатывались в сообществах NLP и информационного поиска и основаны на разложении матриц [Levy and Goldberg, 2014]. Алгоритмы погружения слов обсуждаются в разделе 10.4.

Важное преимущество обучения погружений слов на больших объемах неаннотированных данных состоит в том, что оно дает векторные представления слов, не

[1] Такая интерпретация создания вспомогательных задач из необработанного текста навеяна работой Ando and Zhang [2005a,b].

встречавшихся в обучающем наборе. В идеале представления таких слов должны быть похожи на представления родственных слов, которые встречаются в обучающем наборе, что позволило бы модели лучше обобщаться на незнакомые события. Поэтому желательно, чтобы сходство между векторами слов, обученными алгоритмом без учителя, улавливало именно те аспекты сходства, которые полезны при выполнении стоящей перед сетью задачи.

Можно предположить, что выбор вспомогательной задачи (что предсказывается, на основе какого контекста) оказывает гораздо большее влияние на результирующие векторы, чем использованный метод обучения. В разделе 10.5 приведен обзор различных способов выбора вспомогательной задачи.

Погружения слов, построенные алгоритмами обучения без учителя, имеют и другие применения в NLP, помимо использования для инициализации слоя погружения в нейросетевой модели. Они обсуждаются в главе 11.

10.3.1. Использование предобученных погружений

При использовании предобученных погружений необходимо принять несколько решений. Первое касается предобработки: должны ли предобученные векторы слов использоваться как есть или их нужно нормировать на единичную длину? Ответ зависит от задачи. Во многих алгоритмах погружения слов норма вектора слова коррелирует с частотой этого слова. Нормировка векторов на единичную длину убирает информацию о частоте. Иногда это желательная унификация, а иногда – плачевная потеря информации.

Второе решение касается дополнительной настройки предобученных векторов под конкретную задачу. Рассмотрим матрицу погружений $\boldsymbol{E} \in \mathbb{R}^{|V| \times d}$, которая ассоциирует слова из словаря V с d-мерными векторами. Часто элементы $\boldsymbol{E}$ трактуются как параметры модели и изменяются вместе с остальной сетью. Хотя такой подход неплохо работает, у него есть потенциально нежелательный эффект – представления слов, встречающихся в обучающих данных, изменяются, а других слов нет, даже если они и были близки в оригинальных предобученных векторах. Это может негативно отразиться на обобщаемости, которой мы пытались добиться с помощью процедуры предобучения. Альтернатива – зафиксировать предобученные векторы. Тогда обобщаемость сохраняется, но модель теряет способность адаптировать представления к конкретной задаче. Золотая середина – не трогать $\boldsymbol{E}$, но ввести дополнительную матрицу $\boldsymbol{T} \in \mathbb{R}^{d \times d}$. Вместо того чтобы брать строки $\boldsymbol{E}$, мы берем строки преобразованной матрицы $\boldsymbol{E}' = \boldsymbol{ET}$. Матрица $\boldsymbol{T}$ обучается вместе с сетью, что позволяет точно настроить некоторые аспекты предобученных векторов под задачу. Однако зависящие от задачи адаптации имеют вид линейных преобразований, которые применяются ко всем словам, а не только к встречавшимся на этапе обучения. Недостаток этого подхода в том, что он не позволяет изменить представления только избранных слов (например, если словам *hot* и *cold* были сопоставлены очень похожие векторы, то линейному преобразованию $\boldsymbol{T}$ было бы крайне трудно разделить их). Еще один вариант – оставить $\boldsymbol{E}$ без изменения, но использовать дополнительную матрицу $\boldsymbol{\Delta} \in \mathbb{R}^{|V| \times d}$ и в качестве матрицы погружений взять $\boldsymbol{E}' = \boldsymbol{E} + \boldsymbol{\Delta}$ или $\boldsymbol{E}' = \boldsymbol{ET} + \boldsymbol{\Delta}$. Матрица $\boldsymbol{\Delta}$ инициализируется нулями и обучается вместе с сетью, что позволяет обучиться дополнительным изменениям конкретных слов. Включение большого регуляризирующего штрафа за из-

менение Δ будет способствовать тому, чтобы и после дополнительной настройки представления оставались близки к исходным[1].

10.4. Алгоритмы погружения слов

Нейросетевое сообщество по традиции мыслит в терминах *распределенных представлений* [Hinton et al., 1987]. В отличие от локальных представлений, в которых сущности представляются дискретными символами, а взаимодействия между сущностями кодируются в виде множества дискретных отношений между символами, образующих граф, в распределенных представлениях каждая сущность представляется вектором значений («паттерном активаций»), а смысл сущности и ее связи с другими сущностями улавливаются активациями в этом векторе и сходством между различными векторами. В контексте лингвистической обработки это означает, что слова (и предложения) следует отображать не на дискретные измерения, а в общее пространство низкой размерности, в котором с каждым словом ассоциирован *d*-мерный вектор, а смысл слова определяется его связями с другими словами и значениями активации элементов его вектора.

Специалисты по NLP привыкли мыслить в терминах *дистрибутивной семантики*, когда смысл слова можно вывести из его распределения в корпусе, т. е. из совокупности контекстов, в которых это слово встречается. У слов, встречающихся в похожих контекстах, часто и смысл похож.

Эти два подхода к представлению слов – в терминах паттернов активаций, обученных в процессе выполнения объемлющего алгоритма, и в терминах паттернов совместной встречаемости с другими словами или синтаксическими конструкциями – привели к разработке двух, на первый взгляд очень разных, семейств алгоритмов и направлений исследований.

В разделе 10.4.1 мы рассмотрим дистрибутивный подход к представлению слов, а в разделе 10.4.2 – распределенные представления. В разделе 10.4.3 мы объединим оба мира и покажем, что в современных распределенных представлениях слов большая часть трудной работы выполняется с помощью дистрибутивных сигналов и что между обоими семействами алгоритмов имеются глубокие связи.

10.4.1. Дистрибутивная гипотеза и представления слов

Дистрибутивная гипотеза о языке и смысле слов утверждает, что слова, встречающиеся в одинаковых контекстах, имеют похожий смысл [Harris, 1954]. Эту идею популяризировал Фёрс (Firth [1957]), который говорил «по окружению его узнаете слово». Интуитивно понятно, что, встретив незнакомое слово в предложении, например слово *wampimuk* в предложении *Marco saw a hairy little wampinuk crouching behind a tree* (Марко увидел маленького волосатого вампимука, припавшего к земле за деревом), человек пытается вывести его смысл из контекста. Эта идея положила начало *дистрибутивной семантике* – дисциплине, стремящейся количественно описать сходство между лингвистическими единицами, исходя из

[1] Отметим, что все обновления в процессе градиентного обучения аддитивны, поэтому без регуляризации как обновление $\boldsymbol{E}$ в процессе обучения, так и фиксация $\boldsymbol{E}$, но обновление и поиск в $\boldsymbol{E} + \Delta$ приведут к одинаковым окончательным погружениям. Эти методы дают разные результаты только при наличии регуляризации.

их дистрибутивных свойств в большом корпусе текстов. За обсуждением лингвистических и философских основ дистрибутивной гипотезы отсылаем читателя к работе Sahlgren [2008].

Матрицы слово–контекст

В NLP имеется длинная череда работ[1], в которых дистрибутивные свойства слов изучаются с помощью матриц слово–контекст, в которых i-я строка представляет слово, j-й столбец – лингвистический контекст, в котором могут встречаться слова, а элемент $M_{[i,j]}$ оценивает силу ассоциации между словом и контекстом в большом корпусе. Иными словами, каждое слово представляется разреженным вектором в пространстве большой размерности, кодирующим взвешенный мешок контекстов, в котором это слово встречается. В зависимости от определения контекста и способа измерения силы ассоциации между словом и контекстом возникают различные представления слов. Существуют различные функции для измерения расстояния между векторами слов, которое интерпретируется как семантическое расстояние между ассоциированными с ними словами.

Формально обозначим $\boldsymbol{V_W}$ множество слов (словарь), а $\boldsymbol{V_C}$ – множество возможных контекстов. Будем считать, что все слова и контексты индексированы, так что w_i – i-е слово в словаре, а c_j – j-й элемент множества контекстов. Матрицей слово–контекст называется матрица $\boldsymbol{M}^f \in \mathbb{R}^{|\boldsymbol{VW}|\times|\boldsymbol{VC}|}$, в которой $\boldsymbol{M}^f_{[i,j]} = f(w_i, c_j)$, где функция f измеряет силу ассоциации между словом и контекстом.

Меры сходства

Представив слова векторами, мы можем вычислять сходство между словами как сходство между соответствующими векторами. Весьма распространенной и эффективной мерой является *косинусное расстояние*, равное косинусу угла между векторами:

$$\mathrm{sim}_{\cos}(\boldsymbol{u},\boldsymbol{v}) = \frac{\boldsymbol{u}\cdot\boldsymbol{v}}{\|\boldsymbol{u}\|_2\,\|\boldsymbol{v}\|_2} = \frac{\sum_i \boldsymbol{u}_{[i]}\cdot\boldsymbol{v}_{[i]}}{\sqrt{\sum_i (\boldsymbol{u}_{[i]})^2}\sqrt{\sum_i (\boldsymbol{v}_{[i]})^2}}. \tag{10.1}$$

Популярен также коэффициент сходства Жаккара[2]:

$$\mathrm{sim}_{\text{Jacaard}}(\boldsymbol{u},\boldsymbol{v}) = \frac{\sum_i \min(\boldsymbol{u}_{[i]},\boldsymbol{v}_{[i]})}{\sum_i \max(\boldsymbol{u}_{[i]},\boldsymbol{v}_{[i]})}. \tag{10.2}$$

Веса пар слово–контекст и поточечная взаимная информация

Функция f обычно основана на счетчиках из большого корпуса. Обозначим $\#(w, c)$ количество вхождений слова w в контекст c в корпусе D, а $|D|$ – размер корпуса ($|D| = \sum_{w'\in V_W,\, c'\in V_C} \#(w', c')$. Интуитивно хочется определить $f(w, c)$ как счетчик $f(w, c) = \#(w, c)$ или как нормированный счетчик $f(w, c) = P(w, c) = \#(w, c)/|D|$. Однако при этом получается нежелательный эффект – высокие веса назначаются па-

---

[1] *См.* обзоры Turney and Pantel [2010] и Baroni and Lenci [2010].

[2] Если рассматривать $\boldsymbol{u}$ и $\boldsymbol{v}$ как множества, то коэффициент сходства Жаккара определяется как $\frac{|\boldsymbol{u}\cap\boldsymbol{v}|}{|\boldsymbol{u}\cup\boldsymbol{v}|}$.

рам слово–контекст, относящимся к очень частым контекстам. (Например, будем считать контекстом слова предыдущее слово. Тогда для слова *cat* события *the cat* и *a cat* получат гораздо более высокие оценки, чем *cute cat* и *small cat*, хотя последние намного информативнее). Для подавления этого эффекта лучше определить f так, чтобы предпочтение отдавалось информативным контекстам данного слова – контекстам, в которые данное слово входит чаще, чем другие слова. Эффективная метрика, улавливающая такое поведение, называется *поточечной взаимной информацией* (pointwise mutual information – PMI), это теоретико-информационная мера ассоциации между парой дискретных выходов x и y, определенная следующим образом:

$$\mathrm{PMI}(x,y) = \log \frac{P(x,y)}{P(x)P(y)}. \tag{10.3}$$

В нашем случае PMI(w, c) измеряет ассоциацию между словом w и контекстом c, вычисляемую как логарифм отношения их совместной вероятности (частоты их совместной встречаемости) к произведению безусловных вероятностей (частоты встречаемости по отдельности). PMI можно эмпирически оценить, рассмотрев фактическое количество наблюдений в корпусе:

$$f(w,c) = \mathrm{PMI}(w,c) = \log \frac{\#(w,c)\cdot|D|}{\#(w)\cdot\#(c)}, \tag{10.4}$$

где $\#(w) = \sum_{c' \in V_C} \#(w, c')$ и $\#(c) = \sum_{w' \in V_W} \#(w', c)$ – соответственно частоты w и c в корпусе. Впервые PMI в качестве меры ассоциации в NLP стала использоваться в работе Church and Hanks [1990], а затем нашла широкое применение в задачах определения сходства слов и дистрибутивной семантике [Dagan et al., 1994, Turney, 2001, Turney and Pantel, 2010].

Работа с матрицей PMI сопряжена с вычислительными сложностями. Строки $\boldsymbol{M}^{\mathrm{PMI}}$ содержат много пар слово–контекст (w, c), которые никогда не встречались в корпусе, поэтому для них $\mathrm{PMI}(w, c) = \log 0 = -\infty$. Стандартное решение – использовать в качестве метрики *положительную PMI* (PPMI), в которой все отрицательные значения заменены нулями[1]:

$$\mathrm{PPMI}(w, c) = \max(\mathrm{PMI}(w, c), 0). \tag{10.5}$$

Систематическое сравнение различных схем взвешивания элементов матрицы слово–контекст показывает, что метрика PMI и в еще большей степени PPMI

[1] Если говорить о представлении слов, то за игнорированием отрицательных значений стоит интуитивно понятное соображение: люди легко воспринимают *положительные* ассоциации (например, «Канада» и «снег»), но с гораздо большим трудом – отрицательные («Канада» и «пустыня»). Отсюда следует, что на воспринимаемое сходство двух слов сильнее влияет общий положительный контекст, чем общий отрицательный. Поэтому интуитивно имеет смысл отбрасывать отрицательные контексты, помечая их как «неинформативные» (0). Заметное исключение – случай синтаксического сходства. Например, всем глаголам свойственна очень сильная отрицательная ассоциация с предшествующими детерминативами, а глаголам в прошедшем времени – очень сильная отрицательная ассоциация с предшествующим глаголом «be» и модельными глаголами.

дают наилучшие результаты для широкого круга задач, связанных с определением сходства слов [Bullinaria and Levy, 2007, Kiela and Clark, 2014].

Недостаток PMI состоит в склонности назначать высокое значение редким событиям. Например, если два события встречаются только по одному разу, но вместе, то они получат высокую оценку PMI. Поэтому рекомендуется при использовании PMI задавать пороговый счетчик или еще как-то понижать оценки редких событий.

Понижение размерности посредством разложения матрицы

Потенциальное препятствие представлению слов в виде явного множества контекстов, в которых они встречаются, – разреженность данных: некоторые элементы матрицы $\boldsymbol{M}$ могут быть некорректны, потому что мы не наблюдали достаточного количества примеров. Кроме того, размерность явных векторов слов очень высока (в зависимости от определения контекста количество возможных контекстов может исчисляться сотнями тысяч, а то и миллионами).

Обе проблемы можно сгладить, рассмотрев представление данных низкого ранга, получаемое каким-либо методом *понижения размерности*, например *сингулярным разложением* (SVD).

Метод SVD сводится к разложению матрицы $\boldsymbol{M} \in \mathbb{R}^{|V_W| \times |V_C|}$ в произведение двух узких матриц: матрицы слов $\boldsymbol{W} \in \mathbb{R}^{|V_W| \times d}$ и матрицы контекстов $\boldsymbol{C} \in \mathbb{R}^{|V_C| \times d}$, так что матрица $\boldsymbol{W}\boldsymbol{C}^{\top} = \boldsymbol{M'} \in \mathbb{R}^{|V_W| \times |V_C|}$ является наилучшей аппроксимацией $\boldsymbol{M}$ ранга d в том смысле, что не существует матрицы ранга d, которая была бы ближе к $\boldsymbol{M}$, чем $\boldsymbol{M'}$, в смысле метрики L_2.

Представление $\boldsymbol{M'}$ низкого ранга можно рассматривать как «сглаженный» вариант $\boldsymbol{M}$: на основании устойчивых паттернов, наблюдаемых в данных, некоторые измерения «зафиксированы». Одно из проявлений этого заключается в добавлении слов в контекст, в котором они не встречались, если они совместно встречались с другими словами, находящимися в этом контексте. Кроме того, матрица $\boldsymbol{W}$ позволяет представить каждое слово плотным d-мерным вектором вместо разреженного $|V_C|$-мерного, где $d \ll |V_C|$ (типичный диапазон значений $50 < d < 300$), так что d-мерные векторы улавливают наиболее важные направления изменения в исходной матрице. Затем можно вычислять сходство на основе плотных d-мерных векторов, а не разреженных векторов гораздо большей размерности.

МАТЕМАТИКА SVD Сингулярное разложение (SVD) – это алгебраический метод разложения вещественной или комплексной матрицы $\boldsymbol{M}$ размера $m \times n$ в произведение трех матриц:

$$\boldsymbol{M} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V},$$

где $\boldsymbol{U}$ – вещественная или комплексная матрица размера $m \times m$, $\boldsymbol{D}$ – вещественная или комплексная матрица размера $m \times n$, а $\boldsymbol{V}$ – матрица размера $n \times n$. Матрицы $\boldsymbol{U}$ и $\boldsymbol{V}^{\top}$ *ортонормированные*, т. е. составляющие их векторы-строки взаимно ортогональны и имеют единичную длину. Матрица $\boldsymbol{D}$ диагональная, и элементы на главной диагонали являются сингулярными значениями $\boldsymbol{M}$, расположенными в порядке убывания.

Это разложение точное. У SVD имеется много применений – в машинном обучении и других областях. Мы используем SVD для *понижения размерности* – нахождения такого представления данных высокой размерности в пространстве более низкой размерности, которое сохраняет большую часть информации, содержащейся в исходных данных.

Рассмотрим произведение $\boldsymbol{U\tilde{D}V}$, где $\tilde{\boldsymbol{D}}$ получается из $\boldsymbol{D}$ заменой нулями всех диагональных элементов, кроме первых k. Теперь можно обнулить все строки $\boldsymbol{U}$ и столбцы $\boldsymbol{V}$, кроме первых k, поскольку они все равно обнулятся в результате умножения. После удаления строк и столбцов у нас остаются три матрицы: $\tilde{\boldsymbol{U}}$ ($m\times k$), $\tilde{\boldsymbol{D}}$ ($k\times k$, диагональная) и $\tilde{\boldsymbol{V}}$ ($k\times n$). Произведение

$$\boldsymbol{M'} = \tilde{\boldsymbol{U}}\tilde{\boldsymbol{D}}\tilde{\boldsymbol{V}}$$

является матрицей $m\times n$ *ранга* k.

Матрица $\boldsymbol{M'}$ – произведение узких матриц ($\tilde{\boldsymbol{U}}$ и $\tilde{\boldsymbol{V}}$, где k намного меньше m и n), его можно интерпретировать как *низкоранговую аппроксимацию* $\boldsymbol{M}$.

По *теореме Эккарта–Янга* [Eckart and Young, 1936], матрица $\boldsymbol{M'}$ является *наилучшей k-ранговой аппроксимацией* $\boldsymbol{M}$ относительно потери по норме L_2. Это значит, что $\boldsymbol{M'}$ доставляет минимум норме:

$$\boldsymbol{M}' = \underset{\boldsymbol{X}\in\mathbb{R}^{m\times n}}{\operatorname{argmin}} \left\|\boldsymbol{X}-\boldsymbol{M}\right\|_2, \text{ при условии что } \boldsymbol{X} \text{ имеет ранг } k.$$

Матрицу $\boldsymbol{M'}$ можно рассматривать как сглаженный вариант $\boldsymbol{M}$ в том смысле, что в ней используется только k наиболее важных направлений в данных.

Аппроксимация строк. Низкоразмерные строки матрицы $\boldsymbol{E} = \tilde{\boldsymbol{U}}\tilde{\boldsymbol{D}}$ являются низкоранговыми аппроксимациями высокоразмерных строк исходной матрицы $\boldsymbol{M}$ в том смысле, что вычисление скалярного произведения строк $\boldsymbol{E}$ *эквивалентно* вычислению скалярного произведения строк реконструированной матрицы $\boldsymbol{M'}$, т. е. $\boldsymbol{E}_{[i]}\cdot\boldsymbol{E}_{[j]} = \boldsymbol{M'}_{[i]}\cdot\boldsymbol{M'}_{[j]}$.

Чтобы понять, почему это так, рассмотрим матрицу $m\times m$ $\boldsymbol{S^E} = \boldsymbol{EE}^\top$. Элемент $[i, j]$ этой матрицы равен скалярному произведению строк i и j матрицы $\boldsymbol{E}$: $\boldsymbol{S^E_{[i,j]}} = \boldsymbol{E}_{[i]}\cdot\boldsymbol{E}_{[j]}$. И аналогично для матрицы $\boldsymbol{S^{M'}} = \boldsymbol{M'M'}^\top$.

Мы покажем, что $\boldsymbol{S^E} = \boldsymbol{S^{M'}}$. Напомним, что $\tilde{\boldsymbol{V}}\tilde{\boldsymbol{V}}^\top = \boldsymbol{I}$, поскольку $\tilde{\boldsymbol{V}}$ ортонормированная. Имеем:

$$\begin{aligned}\boldsymbol{S^{M'}} = \boldsymbol{M'M'}^\top &= (\tilde{\boldsymbol{U}}\tilde{\boldsymbol{D}}\tilde{\boldsymbol{V}})(\tilde{\boldsymbol{U}}\tilde{\boldsymbol{D}}\tilde{\boldsymbol{V}})^\top = (\tilde{\boldsymbol{U}}\tilde{\boldsymbol{D}}\tilde{\boldsymbol{V}})(\tilde{\boldsymbol{V}}^\top\tilde{\boldsymbol{D}}^\top\tilde{\boldsymbol{U}}^\top) = \\ &= (\tilde{\boldsymbol{U}}\tilde{\boldsymbol{D}})(\tilde{\boldsymbol{V}}\tilde{\boldsymbol{V}}^\top)(\tilde{\boldsymbol{D}}^\top\tilde{\boldsymbol{U}}^\top) = (\tilde{\boldsymbol{U}}\tilde{\boldsymbol{D}})(\tilde{\boldsymbol{U}}\tilde{\boldsymbol{D}})^\top = \boldsymbol{EE}^\top = \boldsymbol{S^E}.\end{aligned}$$

Таким образом, мы можем использовать строки $\boldsymbol{E}$ вместо высокоразмерных строк $\boldsymbol{M'}$ (и вместо высокоразмерных строк $\boldsymbol{M}$. Аналогичное рассуждение показывает, что можно использовать строки $(\tilde{\boldsymbol{D}}\tilde{\boldsymbol{V}})^\top$ вместо столбцов $\boldsymbol{M'}$).

Когда мы применяем SVD для определения сходства слов, строки $\boldsymbol{M}$ соответствуют словам, столбцы – контекстам, а векторы, образующие строки $\boldsymbol{E}$, – это низкоразмерные представления слов. На практике часто предпочтительнее использовать не матрицу $\boldsymbol{E} = \tilde{\boldsymbol{U}}\tilde{\boldsymbol{D}}$, а более «сбалансированную» матрицу $\boldsymbol{E} = \tilde{\boldsymbol{U}}\sqrt{\tilde{\boldsymbol{D}}}$ или даже вовсе игнорировать сингулярные значения $\tilde{\boldsymbol{D}}$ и брать $\boldsymbol{E} = \tilde{\boldsymbol{U}}$.

10.4.2. От нейросетевых языковых моделей к распределенным представлениям

Специалисты по нейронным сетям отдают предпочтение не описанным выше *методам на основе подсчета*, а использованию *распределенных представлений* смысла слов. В распределенном представлении с каждым словом ассоциирован вектор из $\mathbb{R}^d$, а «смысл» слова в конкретной задаче улавливается различными измерениями этого вектора, а также измерениями других слов. В отличие от явного дистрибутивного представления, где каждое измерение соответствует определенному контексту, в котором встречается слово, измерения распределенного представления не допускают интерпретации, и отдельные измерения необязательно соответствуют каким-то конкретным концепциям. Из распределенной природы представления вытекает, что данный аспект смысла может улавливаться комбинацией многих измерений (быть распределен по измерениям) и что любое измерение может вносить вклад в улавливание различных аспектов смысла[1].

Рассмотрим сеть языкового моделирования, описываемую уравнением (9.3) в главе 9. Контекстом слова является k-грамма предшествующих ему слов. С каждым словом ассоциирован вектор, и конкатенация этих векторов кодируется d_{hid}-мерным вектором $\boldsymbol{h}$ с применением нелинейного преобразования. Затем вектор $\boldsymbol{h}$ умножается на матрицу $\boldsymbol{W}^2$, столбцы которой соответствуют словам, а взаимодействия между $\boldsymbol{h}$ и столбцами $\boldsymbol{W}^2$ определяют вероятности различных слов при условии контекста. Столбцы $\boldsymbol{W}^2$ (а также строки матрицы погружений $\boldsymbol{E}$) – это распределенные представления слов: процесс обучения находит хорошие значения погружений, которые порождают правильные оценки вероятности слова в контексте k-граммы и улавливают «смысл» слова в ассоциированном с ним столбце $\boldsymbol{W}^2$.

Метод Коллоберта–Уэстона

При проектировании сети, описываемой уравнением (9.3), мы имели в виду задачу языкового моделирования, которая выдвигает два важных требования: необходимость порождать *распределения вероятностей* слов и необходимость обусловливания контекстами, которые можно объединять с помощью цепного правила исчисления вероятностей для получения оценок вероятностей на уровне предложения. Необходимость порождать распределения вероятностей диктует необходимость вычислять дорогостоящий нормировочный коэффициент, учитывающий все слова в выходном словаре, а требование разлагать в произведение согласно цепному правилу ограничивает множество обусловливающих контекстов только предшествующими k-граммами.

Если нас интересуют только результирующие представления, то оба ограничения можно ослабить, как сделано в работе Collobert and Weston [2008]. Впоследствии предложенная ими модель была уточнена и представлена более полно в работе Bengio et al. [2009]. Первое изменение, внесенное Коллобертом и Уэстоном,

[1] Заметим, что во многих отношениях явное дистрибутивное представление также является «распределенным»: различные аспекты смысла слова улавливаются группами контекстов, в которых слово встречается, а один контекст может вносить вклад в различные аспекты смысла. К тому же после понижения размерности матрицы слово–контекст отдельные измерения уже не допускают простой интерпретации.

заключалось в изменении контекста слова: вместо предшествующей k-граммы (слов, расположенных левее) предлагалось брать окружающее окно слов (т. е. вычислять $P(w_3|w_1w_2\square w_4w_5)$ вместо $P(w_5|w_1w_2w_3w_4\square)$. Обобщение на другие виды контекстов фиксированного размера $c_{1:k}$ очевидно.

Кроме того, Коллоберт и Уэстон предложили отказаться от требования о вероятностях на выходе. Вместо того чтобы вычислять распределение вероятностей целевых слов при условии контекста, модель всего лишь пытается назначить каждому слову оценку, чтобы правильные слова оценивались выше неправильных. Это избавляет от необходимости выполнять вычислительно дорогостоящую нормировку по выходному словарю, так что время вычислений оказывается независимым от размера выходного словаря. В результате сеть не только гораздо быстрее обучается и работает, но также масштабируется на словари практически неограниченного размера (увеличение словаря приводит лишь к линейному росту объема потребной памяти).

Обозначим w целевое слово, $c_{1:k}$ – упорядоченный список контекстных элементов, а $v_w(w)$ и $v_c(c)$ – функции погружения, которые отображают индексы слов и контекстов в d_{emb}-мерные векторы (начиная с этого момента, будем считать, что размерности векторов слов и контекстов одинаковы). Модель Коллоберта–Уэстона вычисляет оценку $s(w, c_{1:k})$ пары слово–контекст, конкатенируя погружения слова и контекста в один вектор $\boldsymbol{x}$, который подается на вход МСП с одним скрытым слоем, единственным выходом которого является оценка, назначенная этой паре:

$$\begin{aligned} s(w, c_{1:k}) &= g(\boldsymbol{x}\boldsymbol{U})\cdot\boldsymbol{v}; \\ \boldsymbol{x} &= [v_c(c_1); \ldots; v_c(c_k); v_w(w)]; \\ \boldsymbol{U} &\in \mathbb{R}^{(k+1)d_{\text{emb}}\times d_{\text{h}}}, \quad \boldsymbol{v} \in \mathbb{R}^{d_{\text{h}}}. \end{aligned} \tag{10.6}$$

При обучении сети используется потеря ранжирования с зазором, чтобы разность между оценкой правильной пары слово–контекст $(w, c_{1:k})$ и неправильной пары $(w', c_{1:k})$ была не меньше 1. Потеря $L(w, c_{1:k})$ на данной паре слово–контекст вычисляется по формуле:

$$L(w, c, w') = \max(0, 1 - (s(w, c_{1:k}) - s(w', c_{1:k}))), \tag{10.7}$$

где w' – слово, случайно выбранное из словаря. Процедура обучения перебирает пары слово–контекст из корпуса и на каждой итерации выбирает случайное слово w', вычисляет потерю $L(w, c, w')$ на w' и обновляет параметры $\boldsymbol{U}$, $\boldsymbol{v}$, а также погружения слов и контекстов, стремясь минимизировать потерю.

Использование случайно выбранных слов для порождения *отрицательных примеров* некорректных пар слово–контекст с целью управления оптимизацией является также сердцевиной алгоритма Word2Vec, описываемого ниже.

Word2Vec

Очень популярный алгоритм Word2Vec был разработан Томашем Миколовым с коллегами и описан в серии статей [Mikolov et al., 2013b,a]. Как и алгоритм Коллоберта–Уэстона, Word2Vec начинает работу с нейросетевой языковой модели и модифицирует ее с целью ускорить получение результатов. Word2Vec – это не просто алгоритм, а программный пакет, реализующий два разных представления контекста (CBOW и skip-граммы) с двумя разными целевыми функциями

(Negative-Sampling и иерархической Softmax). Здесь мы рассмотрим функцию Negative-Sampling (NS).

Как и в алгоритме Коллоберта–Уэстона, идея варианта NS из Word2Vec заключается в том, чтобы обучить сеть отличать «хорошие» пары слово–контекст от «плохих». Однако Word2Vec заменяет целиком функцию ранжирования с зазором вероятностной. Рассмотрим множество D правильных пар слово–контекст и множество $\bar{D}$ неправильных пар. Цель алгоритма – оценить вероятность $P(D = 1|w, c)$ того, что пара слово–контекст взята из множества D. Эта вероятность должна быть велика (1) для пар из D и мала (0) для пар из $\bar{D}$. Из самого определения вероятности вытекает, что $P(D = 1|w, c) = 1 - P(D = 0|w, c)$. Функция вероятности моделируется сигмоидой от оценки $s(w, c)$:

$$P(D = 1|w, c) = \frac{1}{1 + e^{-s(w,c)}}. \tag{10.8}$$

Глобальная для всего корпуса цель алгоритма состоит в том, чтобы максимизировать логарифмическое правдоподобие данных $D \cup \bar{D}$:

$$L(\Theta; D, \bar{D}) = \sum_{(w,c)\in D} \log P(D = 1|w, c) + \sum_{(w,c)\in \bar{D}} \log P(D = 0|w, c). \tag{10.9}$$

Положительные примеры D выбираются из корпуса. Отрицательные примеры $\bar{D}$ можно сгенерировать разными способами. В Word2Vec для этого применяется следующий процесс: для каждой хорошей пары $(w, c) \in D$ выбрать k слов $w_{1:k}$ и добавить каждую пару (w_i, c) в $\bar{D}$ в качестве отрицательного примера. В результате множество отрицательных примеров $\bar{D}$ оказывается в k раз больше D. Число k является параметром алгоритма.

Отрицательные слова w можно выбирать в соответствии с их частотой в корпусе $\#(w)/\sum_{w'} \#(w')$ или, как сделано в реализации Word2Vec, в соответствии со сглаженным вариантом, когда счетчики возводятся в степень ¾ перед нормировкой: $\#(w)^{0.75}/\sum_{w'} \#(w')^{0.75}$. Второй вариант назначает больший вес менее частым словам и на практике лучше улавливает сходство слов.

CBOW Помимо замены, основанной на зазоре целевой функции на вероятностную, Word2Vec также заметно упрощает определение функции оценивания пар слово–контекст, $s(w, c)$. Для контекста, содержащего несколько слов $c_{1:k}$, в варианте CBOW алгоритма Word2Vec контекстный вектор $\boldsymbol{c}$ определяется как сумма векторов погружений компонент контекста: $\boldsymbol{c} = \sum_{i=1}^{k} \boldsymbol{c}_i$ Затем оценка вычисляется просто как $s(w, c) = \boldsymbol{w} \cdot \boldsymbol{c}$, что дает:

$$P(D = 1|w, c_{1:k}) = \frac{1}{1 + e^{-(\boldsymbol{w}\cdot\boldsymbol{c}_1 + \boldsymbol{w}\cdot\boldsymbol{c}_2 + \ldots + \boldsymbol{w}\cdot\boldsymbol{c}_k)}}.$$

В варианте CBOW теряется информация о порядке следования элементов контекста. Но взамен он позволяет использовать контексты переменной длины. Заметим, однако, что для контекстов ограниченной длины CBOW все же может сохранить информацию о порядке, включив относительную позицию в сам элемент контекста, т. е. назначив разные векторы погружения элементам контекста, занимающим разные относительные позиции.

SKIP-ГРАММЫ В варианте вычисления оценки со skip-граммами зависимость между элементами контекста разрывается еще сильнее. Для k-элементного контекста $c_{1:k}$ предполагается, что элементы c_i не зависят друг от друга. По существу, они рассматриваются как k различных контекстов, т. е. пара слово–контекст $(w, c_{i:k})$ будет представлена в D k контекстами: (w, c_1), ..., (w, c_k). Функция оценивания $s(w, c)$ определяется как в варианте CBOW, но теперь каждый контекст является одним вектором погружения:

$$\begin{aligned} P(D=1|w,c_i) &= \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{c}_i}}; \\ P(D=1|w,c_{1:k}) &= \prod_{i=1}^{k} P(D=1|w,c_i) = \prod_{1=i}^{k} \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{c}_i}}; \\ \log P(D=1|w,c_{1:k}) &= \sum_{i=1}^{k} \log \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{c}_i}}. \end{aligned} \tag{10.10}$$

Несмотря на сильные предположения о независимости элементов контекста, skip-граммный вариант очень эффективен на практике и часто применяется.

10.4.3. Объединяя миры

И дистрибутивный метод «с подсчетом», и распределенный «нейросетевой» метод основаны на дистрибутивной гипотезе в том смысле, что пытаются уловить сходство между словами, ориентируясь на сходство контекстов, в которых они встречаются. На самом деле в работе Levy and Goldberg [2014] показано, что связи между обоими мирами гораздо глубже, чем кажется на первый взгляд.

Обучение моделей Word2Vec дает две матрицы погружений, $\boldsymbol{E}^{\boldsymbol{W}} \in \mathbb{R}^{|V_W| \times d_{\text{emb}}}$ и $\boldsymbol{E}^{\boldsymbol{C}} \in \mathbb{R}^{|V_C| \times d_{\text{emb}}}$, представляющие слова и контексты соответственно. Погружения контекстов после обучения отбрасываются, а погружения слов остаются. Но представим, что мы сохранили матрицу погружений контекстов $\boldsymbol{E}^{\boldsymbol{C}}$, и рассмотрим произведение $\boldsymbol{E}^{\boldsymbol{W}} \times \boldsymbol{E}^{\boldsymbol{C}^{\mathrm{T}}} = \boldsymbol{M}' \in \mathbb{R}^{|\boldsymbol{VW}| \times |\boldsymbol{VC}|}$. При таком взгляде Word2Vec вычисляет разложение неявной матрицы слово–контекст $\boldsymbol{M}'$. Из каких элементов состоит $\boldsymbol{M}'$? Элемент $\boldsymbol{M}'_{[w,c]}$ соответствует скалярному произведению векторов погружений слова и контекста $\boldsymbol{w}\cdot\boldsymbol{c}$. Леви и Гольдберг показали, что для комбинации skip-граммных контекстов и целевой функции Negative-Sampling с k отрицательными примерами в выборке глобальная целевая функция достигает минимума, если положить $\boldsymbol{w}\cdot\boldsymbol{c} = \boldsymbol{M}'_{[w,c]} = \text{PMI}(w, c) - \log k$. То есть Word2Vec неявно определяет факторизацию матрицы, тесно связанной с хорошо известной матрицей слово–контекст PMI! Удивительно, что при этом мы обошлись без явного построения матрицы $\boldsymbol{M}'$[1].

[1] Если достигалось оптимальное назначение, то решение, полученное применением skip-грамм с отрицательной выборкой (SGNS), то же, что полученное применением SVD к матрице слово–контекст. Разумеется, из-за низкой размерности d_{emb} векторов $\boldsymbol{w}$ и $\boldsymbol{c}$ может оказаться невозможным удовлетворить соотношение $\boldsymbol{w}\cdot\boldsymbol{c} = \text{PMI}(w, c) - \log k$ для всех пар $\boldsymbol{w}$ и $\boldsymbol{c}$, и процедура оптимизации попытается найти наилучшее достижимое решение, уплачивая штраф за каждое отклонение от оптимального назначения. Именно в этом различаются целевые функции SGNS и SVD: SVD применяет квадратичный штраф за отклонение, а SGNS – более сложный штраф.

В приведенном выше анализе предполагается, что отрицательные примеры выбираются в соответствии с частотой слов в корпусе $P(w) = \#(w)/\sum_{w'} \#(w')$. Напомним, что реализация Word2Vec производит выборку из модифицированного распределения $P^{0.75}(w) = \#(w)^{0.75}/\sum_{w'} \#(w')^{0.75}$. При такой схеме выборки оптимальное значение изменяется на $\text{PMI}^{0.75}(w, c) - \log k = \log \frac{P(w,c)}{P^{0.75}(w)P(c)} - \log k$. На самом деле применение так модифицированной версии PMI при построении разреженного и явного дистрибутивного векторов заодно улучшает и сходство слов.

Алгоритмы Word2Vec очень эффективны на практике и прекрасно масштабируются, что позволяет обучать представления слов на очень больших словарях, содержащих миллиарды слов, всего за несколько часов, предъявляя очень скромные требования к памяти. Связь между SGNS-вариантом Word2Vec и факторизацией матрицы слово–контекст наводит мост между нейросетевыми методами и традиционными методами на основе подсчета, подсказывая, что знания, полученные при изучении «дистрибутивного» представления, можно перенести на «распределенные» алгоритмы и, наоборот, и что на глубинном уровне эти два семейства алгоритмов эквивалентны.

10.4.4. Другие алгоритмы

Существует много вариантов алгоритмов Word2Vec, но ни один не дает представления слов, которое убедительно превосходило бы конкурентов в количественном или качественном отношениях. Ниже перечислено несколько наиболее популярных.

NCE Шумосопоставительное оценивание (noise-contrastive estimation – NCE), описанное в работе Mnih and Kavukcuoglu [2013], очень похоже на вариант SGNS алгоритма Word2Vec, но распределение $P(D = 1|w, c_i)$ моделируется не как в уравнении (10.10), а следующим образом:

$$P(D = 1|w, c_i) = \frac{e^{-\mathbf{w}\cdot\mathbf{c}_i}}{e^{-\mathbf{w}\cdot\mathbf{c}_i} + k \times q(w)}; \tag{10.11}$$

$$P(D = 0|w, c_i) = \frac{k \times q(w)}{e^{-\mathbf{w}\cdot\mathbf{c}_i} + k \times q(w)}, \tag{10.12}$$

где $q(w) = \#(w)/|D|$ – наблюдаемая частота униграммы w в корпусе. Этот алгоритм основан на вероятностной технике моделирования шумосопоставительной оценки [Gutmann and Hyvärinen, 2010]. Согласно работе Levy and Goldberg [2014], эта целевая функция эквивалентна факторизации матрицы слово–контекст, элементами которой являются логарифмы условных вероятностей $\log P(w|c) - \log k$.

GLOVE Алгоритм GloVe [Pennington et al., 2014] строит явную матрицу слово–контекст и обучает вектор слов и контекстов $\boldsymbol{w}$ и $\boldsymbol{c}$, пытаясь удовлетворить соотношению:

$$\boldsymbol{w}\cdot\boldsymbol{c} + \boldsymbol{b}_{[w]} + \boldsymbol{b}_{[c]} = \log\#(w, c), \quad \forall(w, c) \in D, \tag{10.13}$$

где $\boldsymbol{b}_{[w]}$ и $\boldsymbol{b}_{[c]}$ – обученные смещения для слов и контекстов. Процедура оптимизации рассматривает наблюдаемые пары слово–контекст, пропуская события с нулевым

счетчиком. В терминах факторизации матриц, если зафиксировать $\boldsymbol{b}_{[w]} = \log\#(w)$ и $\boldsymbol{b}_{[c]} = \log\#(c)$, мы получим целевую функцию, очень похожую на применяемую для факторизации матрицы слово–контекст PMI, но сдвинутую на $\log(|D|)$. Однако в GloVe эти параметры не фиксируются, а обучаются, что дает еще одну степень свободы. Целевой функцией оптимизации является взвешенная среднеквадратическая потеря, которая назначает больший вес правильной реконструкции частых объектов. Наконец, при использовании одинаковых словарей слов и контекстов модель GloVe предлагает представлять каждое слово в виде суммы соответствующих векторов погружения слова и контекста.

10.5. Выбор контекстов

Выбор контекста, по которому предсказывается слово, оказывает существенное влияние на результирующие векторы слов и кодируемое ими сходство.

В большинстве случаев в качестве контекста слова берутся соседние с ним слова – либо в узком окружающем окне, либо в том же предложении, абзаце или документе. Иногда текст автоматически разбирается синтаксическим анализатором, и контекст выводится из синтаксического соседства, индуцированного деревьями разбора. Бывает, что в определения слова и контекста включаются также части слов, например префиксы или суффиксы.

10.5.1. Подход на основе окон

Самым распространенным является подход на основе скользящих окон, в котором вспомогательные задачи создаются путем рассмотрения последовательности $2m + 1$ слов. Среднее слово называется *фокусным*, а m слов с каждой стороны – *контекстными*. Затем создается либо одна задача, цель которой – предсказать фокусное слово, зная все контекстные (представленные в виде CBOW [Mikolov et al., 2013b] или путем конкатенации векторов [Collobert and Weston, 2008]), либо $2m$ различных задач, в каждой из которых фокусное слово объединяется в пару с одним из контекстных. Подход с $2m$ задачами, популяризированный в работе Mikolov et al. [2013a], называется *skip-граммной* моделью. Показано, что *skip-граммные* подходы надежны, допускают эффективное обучение [Mikolov et al., 2013a, Pennington et al., 2014] и часто дают результаты, не уступающие лучшим из существующих в отрасли.

ВЛИЯНИЕ РАЗМЕРА ОКНА Размер скользящего окна сильно влияет на сходство результирующих векторов. Широкие окна обычно порождают тематическое сходство (например, слова «dog» (собака), «bark» (лаять) и «leash» (поводок) попадут в одну группу, равно как слова «walked» (шел), «run» (бежал) и «walking» (идущий)), тогда как узкие склонны к выявлению функционального и синтаксического схода (т. е. «Poodle» (пудель), «Pitbull» (питбуль) и «Rottweiler» (ротвейлер) или «walking» (идущий), «running» (бегущий) и «approaching» (приближающийся)).

ПОЗИЦИОННЫЕ ОКНА Если используется представление контекста в виде CBOW или skip-грамм, то все контекстные слова в окне обрабатываются одинаково. Не важно, близко находится контекстное слово к фокусному или далеко, до него или после. Такую информацию легко можно учесть с помощью *пози-*

ционных контекстов, когда для каждого контекстного слова указывается также его положение относительно фокусного (контекстное слово «the» превращается в «the:+2», откуда видно, что слово находится на два слова правее фокусного). Использование позиционного контекста в сочетании с меньшим размером окна обычно порождает сходство скорее синтаксического характера, причем в одну группу склонны попадать слова, являющиеся одной частью речи или обладающие функциональным сходством в терминах семантики. В работе Ling et al. [2015a] показано, что позиционные векторы эффективнее оконных, когда применяются для инициализации сетей для частеречной разметки и анализа синтаксических зависимостей.

ВАРИАНТЫ Есть множество вариантов оконного подхода. Можно лемматизировать слова до обучения, применять нормализацию текста, отфильтровывать слишком короткие или слишком длинные предложения или заменять заглавные буквы строчными (*см.*, например, шаги предобработки, описанные в работе dos Santos and Gatti [2014]). Можно сделать подвыборку части корпуса, пропуская с некоторой вероятностью создание задач по окнам, в которых фокусные слова слишком распространенные или слишком редкие. Можно назначать разные веса различным позициям окна, ставя во главу угла правильное предсказание близких, а не разнесенных пар слово–контекст. Все подобные решения – гиперпараметры, которые надо вручную задавать до начала обучения и которые влияют на результирующие векторы. В идеале их следует подстраивать под конкретную решаемую задачу. Своим высоким качеством семейство алгоритмов Word2Vec в немалой степени обязано выбору хороших значений этих гиперпараметров. Некоторые из них (а также ряд других) подробно обсуждаются в работе Levy et al. [2015].

10.5.2. Предложения, абзацы или документы

Применяя skip-граммы (или CBOW), можно в качестве контекста слова рассматривать другие слова, встречающиеся в том же предложении, абзаце или документе. Это эквивалентно использованию очень широких окон в надежде на то, что векторы слов смогут уловить тематическое сходство (слова, относящиеся к одной теме, которые мы ожидаем встретить в одном документе, с большей вероятностью будут представлены похожими векторами).

10.5.3. Синтаксическое окно

В некоторых работах линейный контекст внутри предложения заменяется синтаксическим [Bansal et al., 2014, Levy and Goldberg, 2014]. Текст автоматически разбирается анализатором зависимостей, и в качестве контекста берутся слова, оказавшиеся рядом в дереве разбора, вместе с их синтаксическими связями. При этом порождается *функциональное* сходство, когда в одну группу попадают слова, выполняющие одну и ту же роль в предложении (например, цвета, названия школ, глаголы, обозначающие движение). Имеет место также синтаксическая группировка, когда в одну группу включаются слова с общим словоизменением [Levy and Goldberg, 2014].

ВЛИЯНИЕ КОНТЕКСТА В следующей таблице, взятой из работы Levy and Goldberg [2014], показаны первые пять слов, имеющих наибольшее сходство с некоторыми начальными словами. Обучение производилось на окнах типа мешка слов размера 5 и 2 (BoW5 и BoW2), а также на синтаксических контекстах (Deps) с использованием одного и того же корпуса (Википедия) и алгоритма погружения (Word2Vec).

Обратите внимание, что для некоторых слов (например, *batman*) найденные похожие слова мало зависят от контекста, тогда как в других случаях заметна четкая тенденция: чем шире контекстное окно, тем выше тематическое сходство (*hogwarts* похоже на другие слова из мира Гарри Поттера, *turing* связано с вычислимостью, *dancing* похоже на другие формы этого слова); в то же время синтаксические контексты порождают в большей степени функциональное сходство (*hogwarts* похоже на другие выдуманные и невыдуманные школы, *turing* – на фамилии других ученых, а *dancing* – на другие герундии, относящиеся к развлечениям). Небольшое контекстное окно оказалось где-то посередине между этими крайностями.

Это еще раз подтверждает мысль, что выбор контекста оказывает огромное влияние на получающиеся представления слов, поэтому его необходимо принимать во внимание при использовании погружений слов «без учителя».

Целевое слово	BoW5	BoW5	Deps
batman	nightwing aquaman catwoman superman manhunter	superman superboy aquaman catwoman batgirl	superman superboy supergirl catwoman aquaman
hogwarts	dumbledore hallows half-blood malfoy snape	evernight sunnydale garderobe blandings collinwood	sunnydale collinwood calarts greendale millfield
turing	nondeterministic non-deterministic computability deterministic finite-state	non-deterministic finite-state nondeterministic buchi primality	pauling hotelling heting lessing hamming
florida	gainesville fla jacksonville tampa lauderdale	fla alabama gainesville tallahassee texas	texas louisiana georgia california carolina
object-oriented	aspect-oriented smalltalk event-driven prolog domain-specific	aspect-oriented event-driven objective-c dataflow 4gl	event-driven domain-specific rule-based data-driven human-centered
dancing	singing dance dances dancers tap-dancing	singing dance dances breakdancing clowning	singing rapping breakdancing miming busking

10.5.4. Многоязычные контексты

Еще одна возможность – использование многоязычных контекстов, полученных в результате перевода [Faruqui and Dyer, 2014, Hermann and Blunsom, 2014]. Например, если имеется большой корпус параллельных текстов, выровненных по предложениям, можно прогнать двуязычную модель выравнивания, например IBM model 1 или model 2 (т. е. использовать программу GIZA++), а затем воспользоваться полученными результатами для вывода контекста слов. Здесь под контекстом слова понимаются выровненные с ним слова на другом языке. При таком выравнивании синонимичным словам будут соответствовать похожие векторы. Некоторые авторы работают на уровне выравнивания предложений, не полагаясь на выравнивание слов [Gouws et al., 2015], или обучают сквозную нейронную сеть машинного перевода и используют получившиеся погружения слов. Заманчиво соединить одноязычный подход на основе окон с многоязычным, создавая вспомогательные задачи обоих видов. При этом, вероятно, будут порождены векторы, похожие на те, что получаются при оконном подходе, но уменьшится нежелательный эффект оконного подхода, из-за которого антонимы (например, hot и cold, high и low) зачастую представляются похожими векторам [Faruqui and Dyer, 2014]. Дополнительное обсуждение многоязычного погружения слов и сравнение различных методов *см*. в работе Levy et al. [2017].

10.5.5. Представления на основе литер и подслов

Есть интересное направление работ, связанное с попытками вывести векторное представление слова из составляющих его литер. Такие подходы могут быть особенно полезны в задачах по природе своей синтаксических, когда паттерны литер внутри слов тесно связаны с их синтаксической функцией. Дополнительным преимуществом является очень малый размер модели (нужно хранить по одному вектору для каждой литеры алфавита плюс несколько небольших матриц) и способность порождать вектор погружения для любого встретившегося слова. В работах dos Santos and Gatti [2014], dos Santos and Zadrozny [2014] и Kim et al. [2015] погружение слов моделируется с помощью сверточной сети (*см*. главу 13), построенной по литерам. В работе Ling et al. [2015b] для той же цели используется конкатенация конечных состояний двух кодировщиков РНС (типа LSTM) (глава 14), один из которых читает литеры слева направо, а другой – справа налево. Оба метода дают очень сильные результаты для частеречной разметки. В работе Ballesteros et al. [2015] показано, что такое кодирование с помощью двух LSTM пригодно также для представления слов в задаче разбора синтаксических зависимостей в языках с развитым морфологическим строем.

Интерес к выводу представлений слов из представлений их литер продиктован *проблемой неизвестных слов* – что делать, когда встречается слово, для которого нет вектора погружения? Работа на уровне литер в значительной степени смягчает эту проблему, поскольку словарь возможных литер существенно меньше словаря возможных слов. Но работать на уровне литер очень трудно, поскольку соотношение между формой (литеры) и функцией (синтаксис, семантика) в языке никак не формализовано. Требование работать только на уровне литер может оказаться неоправданно сильным ограничением. Некоторые исследователи предлага-

ют компромиссное решение, когда слово представляется комбинацией вектора самого слова и векторов составляющих его подслов. Тогда погружения подслов помогают обобществлять информацию между различными словами с похожими формами, а также производить откат на уровень подслов, если слово неизвестно. В то же время модель не обязана полагаться только на форму, если имеется достаточно примеров слова. В работе Botha and Blunsom [2014] предлагается моделировать вектор погружения слова в виде суммы вектора, зависящего от этого слова, если таковой имеется, и векторов для различных морфологических частей слова (эти части выводятся с помощью программы Morfessor [Creutz and Lagus, 2007], реализующей метод морфологического разбора без учителя). В работе Gao et al. [2014] предлагается использовать в качестве базовых признаков не только само слово, но и отдельный признак (а стало быть, отдельный вектор погружения) для каждой буквенной триграммы в слове.

Еще один компромисс между литерами и словами заключается в том, чтобы разбить слова на «осмысленные единицы», которые больше литер и автоматически выводятся из корпуса. Один такой подход – использовать кодирование байтовых пар (Byte-Pair Encoding – BPE) [Gage, 1994] – описан в работе Sennrich et al. [2016a] в контексте машинного перевода, где оказался очень эффективным. В этом случае мы заранее задаем размер словаря (например, 10 000), а затем ищем 10 000 единиц, которые могут представить все слова в словаре корпуса, применяя следующий алгоритм, взятый из работы Sennrich et al. [2016a].

> Инициализировать словарь символов словарем литер и представить каждое слово в виде последовательности литер плюс специальный символ конца слова «.», который позволяет восстановить исходную лексемизацию после перевода. Итеративно подсчитывать все пары символов и заменять каждое вхождение самой частой пары (A, B) новым символом AB. Каждая операция слияния порождает новый символ, который представляет литерную *n*-грамму. Частые литерные *n*-граммы (или целые слова) в конечном итоге будут слиты в один символ, так что BPE не требует краткого списка. Окончательный размер словаря символов равен размеру начального словаря плюс количество операций слияния – последнее является единственным гиперпараметром алгоритма. Для большей эффективности мы не рассматриваем пары, пересекающие границу слова. Таким образом, алгоритм можно прогнать для словаря, выделенного из текста, в котором каждому слову сопоставлен вес, определяемый его частотой.

10.6. Обработка многословных единиц и словоизменения

Два еще недостаточно изученных вопроса в части представлений слов связаны с определением слова. В алгоритмах погружения слов без учителя предполагается, что слова соответствуют лексемам (последовательности литер без пробелов и знаков препинания, *см*. обсуждение вопроса «что такое слово?» в разделе 6.1). Это определение часто не годится.

В английском языке полно *многолексемных единиц* типа *New York* или *ice cream*, а также не столь жестко фиксированных словосочетаний, например: *Boston University* или *Volga River*, – которым желательно сопоставить один вектор.

Во многих языках, отличных от английского, развитая система морфологического словоизменения приводит к образованию форм, которые относятся к одной и той же концепции, но выглядят по-разному. Например, во многих языках прилагательные изменяются по числу и роду, т. е. слово *yellow*, описывающее имя существительное мужского рода множественного числа, отличается по написанию от слова *yellow*, описывающего имя существительное женского рода единственного числа. Хуже того, поскольку правила словоизменения диктуют также формы соседних слов (существительные рядом с формой *yellow* женского рода единственного числа сами должны быть женского рода единственного числа), различные флексии одного и того же слова часто оказываются непохожими друг на друга.

Хотя ни у одной из этих проблем нет хорошего решения, обе можно в какой-то степени разрешить посредством детерминированной предобработки текста так, чтобы он лучше отвечал желаемому определению слова.

В случае многолексемных единиц можно построить их список и заменять их в тексте одной сущностью (т. е. заменять все вхождения New York на New_York). В работе Mikolov et al. [2013a] предложен основанный на метрике PMI метод автоматического создания такого списка: рассматривается PMI пары слов, и пары, для которых оценка PMI превышает заданный порог, объединяются. Этот процесс затем повторяется для объединения пары + слово в тройку и т. д. После этого к предобработанному таким способом корпусу применяется алгоритм вычисления погружений. Эта грубая, но эффективная эвристика реализована в пакете Word2Vec и позволяет строить погружения для наиболее распространенных многолексемных единиц[1].

Что касается словоизменения, проблему можно сгладить, выполнив предварительную лемматизацию всех или некоторых слов в корпусе и строя векторы погружения для лемм, а не для флективных форм.

Родственной формой предобработки является частеречная разметка корпуса и замена слова парой (слово, часть речи). При этом могут быть созданы две разные лексемы $book_{NOUN}$ (книга) и $book_{VERB}$ (бронировать), с каждой из которых будет ассоциирован свой вектор погружения. Дополнительные сведения о взаимосвязях морфологического словоизменения и алгоритмов погружения слов *см*. в работах Avraham and Goldberg [2017], Cotterell and Schutze [2015].

10.7. Ограничения дистрибутивных методов

Дистрибутивная гипотеза предлагает привлекательную платформу для установления сходства слов посредством представления слов с учетом контекстов, где они встречаются. Но ей присущи внутренние ограничения, которые следует принимать во внимание при использовании порожденных представлений.

Определение сходства. Сходство в дистрибутивных методах определяется в чисто операционных терминах: слова похожи, если встречаются в похожих кон-

[1] Подробное обсуждение эвристик для нахождения информативных словосочетаний *см*. в работе Manning and Schütze [1999, Chapter 5].

текстах. Но на практике сходство многогранно. Рассмотрим, к примеру, слова *dog* (собака), *cat* (кошка) и *tiger* (тигр). С одной стороны, *cat* больше похоже на *dog*, чем на *tiger*, поскольку оба являются домашними животными. С другой стороны, *cat* можно считать более похожим на *tiger*, поскольку оба относятся к семейству кошачьих. В одних случаях могут быть предпочтительнее одни грани, в других – другие, а бывает так, что текст сильнее свидетельствует в пользу определенных граней. Дистрибутивные методы почти не позволяют контролировать характер индуцированных ими зависимостей. В какой-то степени этим можно управлять путем выбора обусловливающих контекстов (раздел 10.5), но это решение далеко от идеала.

Белые вороны. Если текст используется в роли обусловливающего контекста, то многие относительно «тривиальные» свойства слова могут не найти отражения в тексте, поэтому не войдут и в представление. Это связано со свойственным людям и хорошо описанным особенностям использования языка, проистекающим из желания повысить эффективность общения: человек больше склонен сообщать новую информацию, чем уже известную. Таким образом, если человек говорит о *серой вороне*, то, скорее всего, назовет ее просто *вороной*, а говоря о *белой вороне*, почти наверняка упомянет ее цвет. Из-за этого модель, обученная только на текстовых данных, может оказаться далеко не объективной.

Антонимы. Слова, противоположные по смыслу (*хороший* и *плохой*, *покупать* и *продавать*, *горячий* и *холодный*), часто встречаются в похожих контекстах (предмет, который может быть горячим, может быть и холодным; то, что покупают, обычно и продают). Поэтому модели, основанные на дистрибутивной гипотезе, склонны рассматривать антонимы как очень похожие слова.

Предвзятость корпуса. Хорошо это или плохо, но дистрибутивные методы отражают паттерны использования в корпусе, на котором основаны, а корпусы, в свою очередь, отражают предвзятость людей в реальном мире (культурную и прочую). Действительно, в работе Caliskan-Islam et al. [2016] обнаружено, что дистрибутивные векторы слов кодируют *все документированные в психологии лингвистические предвзятости, которые мы искали*, включая расовые и гендерные стереотипы (т. е. европейские американские имена ближе к приятным словам, а афроамериканские – ближе к неприятным; женские имена чаще ассоциируются со словами, относящимися к семье, чем к карьере; можно предсказать процент женщин, занятых в некоторой профессии согласно данным переписи США, проанализировав векторное представление названия профессии). Как и в случае антонимов, это поведение может быть желательным или нежелательным в зависимости от ситуации: если наша цель – угадать пол человека, то знание о том, что медсестрами, как правило, работают женщины, а врачами – мужчины, может оказаться желательным свойством алгоритма. Но во многих других случаях мы хотели бы игнорировать такую предвзятость. Как бы то ни было, описанные тенденции в индуцированном сходстве слов следует учитывать при использовании дистрибутивных представлений. Более подробное обсуждение *см*. в работах Caliskan-Islam et al. [2016] и Bolukbasi et al. [2016].

Недостаток контекста. Дистрибутивные методы агрегируют контексты, в которых слово встречается в большом корпусе. Результатом является представление слова, *независимое от контекста*. На практике никакого контекстно-независимого смысла слова не существует. Как сказано в работе Firth [1935], «*полный смысл*

слова всегда раскрывается в контексте, и никакое исследование смысла в отрыве от контекста не следует принимать всерьез». Очевидным проявлением этого принципа является *полисемия*: некоторые слова, очевидно, многозначны: слово *банка* может означать как цилиндрический сосуд, так и морскую отмель; слово *звезда* – абстрактную геометрическую фигуру, знаменитость или небесное тело и т. д. Попытка использовать один вектор для всех этих значений сопряжена с проблемами. Помимо проблемы многозначности, есть еще и гораздо более тонкие проблемы вариации смысла слова в зависимости от контекста.

Глава 11

Использование погружений слов

В главе 10 мы обсуждали алгоритмы построения слов векторов по большим массивам неаннотированных данных. Такие векторы могут быть очень полезны для инициализации матриц погружений слов в специализированных нейронных сетях. Но у них есть и самостоятельная практическая ценность вне контекста нейронных сетей. Некоторые применения такого рода мы и обсудим в этой главе.

Обозначения. В этой главе предполагается, что каждому слову сопоставлен целочисленный индекс, а символы w или w_i относятся к слову и его индексу. Символом $\boldsymbol{E}_{[w]}$ обозначается строка матрицы $\boldsymbol{E}$, соответствующая слову w. Иногда мы будем использовать символы $\boldsymbol{w}$, $\boldsymbol{w_i}$ для обозначения векторов, соответствующих словам w и w_i.

11.1. Получение векторов слов

Векторы погружений слов легко обучить на корпусе текстов, а также общедоступны эффективные реализации алгоритмов обучения. Более того, можно даже скачать готовые векторы слов, уже обученные на очень больших текстовых массивах (правда, надо иметь в виду, что особенности режима обучения и самого корпуса могут оказывать сильное влияние на результирующие представления и что имеющиеся предобученные представления необязательно оптимальны для вашего конкретного случая).

На момент написания этой книги эффективные реализации алгоритмов Word2Vec были доступны в виде автономной библиотеки[1], а также Python-пакета GenSim[2]. Имеется также модифицированный двоичный код Word2Vec, позволяющий использовать произвольные контексты[3]. Также доступна эффективная реализация модели GloVe[4]. Предобученные векторы слов для английского языка можно получить от Google[5], Стэнфордского университета[6] и из других источников. Предобученные векторы для прочих языков можно скачать из проекта Polyglot[7].

---

[1] https://code.google.com/archive/p/word2vec/.

[2] https://radimrehurek.com/gensim/.

[3] https://bitbucket.org/yoavgo/word2vecf.

[4] http://nlp.stanford.edu/projects/glove/.

[5] https://code.google.com/archive/p/word2vec/.

[6] http://nlp.stanford.edu/projects/glove/.

[7] http://polyglot.readthedocs.org.

11.2. Сходство слов

Помимо подачи на вход нейронной сети, основное применение предобученных векторов погружений – вычисление сходства между словами с помощью некоторой функции от векторов sim($\boldsymbol{u}$, $\boldsymbol{v}$). Для измерения сходства часто используется эффективный косинусный коэффициент (или коэффициент Отиаи), равный косинусу угла между векторами:

$$\mathrm{sim}_{\cos}(\boldsymbol{u},\boldsymbol{v}) = \frac{\boldsymbol{u}\cdot\boldsymbol{v}}{\|\boldsymbol{u}\|_2 \|\boldsymbol{v}\|_2}. \tag{11.1}$$

Если векторы $\boldsymbol{u}$ и $\boldsymbol{v}$ имеют единичную длину ($\|\boldsymbol{u}\|_2 = \|\boldsymbol{v}\|_2 = 1$), то косинусный коэффициент сводится к скалярному произведению $\boldsymbol{u}\cdot\boldsymbol{v} = \sum_i \boldsymbol{u}_{[i]}\boldsymbol{v}_{[i]}$. Работать со скалярными произведениями очень удобно с вычислительной точки зрения, поэтому принято нормировать матрицы погружений так, чтобы все строки имели единичную длину. Начиная с этого момента, мы будем предполагать, что матрица $\boldsymbol{E}$ нормирована именно таким образом.

11.3. Кластеризация слов

Векторы слов легко кластеризуются с помощью таких алгоритмов, как *KMeans* (*K* средних), определенных на евклидовом пространстве. Затем кластеры можно использовать в качестве признаков в алгоритмах обучения, работающих с дискретными признаками, или в других системах, требующих дискретных символов, например в информационно-поисковых системах индексирования.

11.4. Нахождение похожих слов

Если матрица погружений нормирована по строкам, как описано выше, то косинусный коэффициент двух слов w_1 и w_2 равен:

$$\mathrm{sim}_{\cos}(w_1, w_2) = \boldsymbol{E}_{[w_1]} \cdot \boldsymbol{E}_{[w_2]}. \tag{11.2}$$

Нередко нас интересуют k слов, наиболее похожих на данное. Обозначим $\boldsymbol{w} = \boldsymbol{E}_{[w]}$ вектор, соответствующий слову w. Сходство со всеми остальными словами можно вычислить, умножив матрицу на вектор: $\boldsymbol{s} = \boldsymbol{E}\boldsymbol{w}$. В результате получаем вектор сходств $\boldsymbol{s}$, i-й элемент которого $s_{[i]}$ содержит сходство w с i-м словом из словаря (i-я строка $\boldsymbol{E}$). k наиболее похожих слов можно найти, вычислив индексы, соответствующие k наибольшим элементам $\boldsymbol{s}$.

В современной оптимизированной библиотеке для научных расчетов, например numpy[1], операция умножения матрицы на вектор занимает несколько миллисекунд для матриц погружений, состоящих из сотен тысяч векторов, так что вычисление сходства производится довольно быстро.

Сходство слов, вычисленное на основе дистрибутивных мер, можно сочетать с другими формами сходства. Например, можно определить меру сходства, осно-

[1] http://www.numpy.org/.

ванную на орфографической близости (слова, в которых много одинаковых букв). Выделив из списка первых k дистрибутивно похожих орфографически слов, мы сможем найти варианты написания и типичные опечатки.

11.4.1. Сходство с группой слов

Нас может заинтересовать нахождение слова, наиболее похожего на *группу* слов. Такая необходимость возникает, когда имеется список родственных слов и мы хотим дополнить его (например, есть список четырех стран и требуется включить в него дополнительные страны, или есть список названий генов и требуется найти названия дополнительных генов). Еще одно применение – когда требуется установить сходство со смыслом слова. Создав список слов, связанных с этим смыслом, мы сможем адресовать запрос о сходстве к самому смыслу.

Существует много способов определить, что такое сходство объекта с группой. Мы будем считать, что сходство с группой – это сходство со средним объектом группы, т. е. если дана группа слов $w_{1:k}$, то ее сходство со словом w определяется как $\text{sim}(w, w_{1:k}) = \frac{1}{k}\sum_{i=1}^{k}\text{sim}_{\cos}(\boldsymbol{w}, \boldsymbol{w_i})$.

В силу линейности для вычисления среднего косинусного коэффициента между группой слов и остальными словами также нужно выполнить одно умножение матрицы на вектор, на этот раз матрицы погружений на средний вектор слов в группе. Вектор $\boldsymbol{s}$, в котором $\boldsymbol{s}_{[w]} = \text{sim}(w, w_{1:k})$, вычисляется по формуле:

$$\boldsymbol{s} = \boldsymbol{E}(\boldsymbol{w_1} + \boldsymbol{w_2} + \ldots + \boldsymbol{w_k})/k. \tag{11.3}$$

11.5. Вычеркивание лишних

Рассмотрим проблему белых ворон: дан список слов и требуется найти в нем постороннее слово. Это можно сделать, сравнив сходство каждого слова со средним вектором слов группы и вернув то слово, для которого сходство минимально.

11.6. Сходство коротких документов

Иногда мы хотим вычислить сходство документов. Хотя для достижения наилучших результатов, вероятно, следует воспользоваться специализированными моделями, решения на основе предобученных погружений слов зачастую оказываются вполне конкурентоспособными, особенно если речь идет о таких коротких документах, как веб-запросы, газетные заголовки или твиты. Идея заключается в том, чтобы представить каждый документ в виде мешка слов и определить сходство документов как сумму попарных сходств слов, принадлежащих документам. Формально рассмотрим два документа: $D_1 = w_1^1, w_2^1, \ldots, w_m^1$ и $D_2 = w_1^2, w_2^2, \ldots, w_n^2$ – и определим сходство документов следующим образом:

$$\text{sim}_{\text{doc}}(D_1, D_2) = \sum_{i=1}^{m}\sum_{j=1}^{n}\cos(\boldsymbol{w}_i^1, \boldsymbol{w}_j^2).$$

Применяя простейшие линейно-алгебраические преобразования, можно показать, что для нормированных векторов слов эта функция сходства равна скалярному произведению представлений документов непрерывными мешками слов:

$$\text{sim}_{\text{doc}}(D_1, D_2) = \left(\sum_{i=1}^{m} \boldsymbol{w}_i^1\right) \cdot \left(\sum_{j=1}^{n} \boldsymbol{w}_j^2\right).$$

Рассмотрим набор документов $D_{1:k}$ и обозначим $\boldsymbol{D}$ матрицу, i-я строка которой содержит представление документа D_i в виде непрерывного мешка слов. Тогда сходство между новым документом $D' = w'_{1:n}$ и каждым документом, входящим в набор, можно вычислить с помощью одного умножения матрицы на вектор: $\boldsymbol{s} = \boldsymbol{D} \cdot (\sum_{i=1}^{k} \boldsymbol{w}'_i)$.

11.7. Словесные аналоги

Миколов с коллегами [Mikolov et al., 2013a, Mikolov et al., 2013] сделали интересное наблюдение, которое немало способствовало популярности погружений слов: над векторами слов можно производить «алгебраические операции», получая осмысленные результаты. Например, имея погружения слов, обученные алгоритмом Word2Vec, можно взять вектор слова *king* (король), вычесть из него слово *man* (мужчина), прибавить слово *woman* (женщина), и оказывается, что вектором, ближайшим к результату (если исключить слова king, man и woman), будет вектор, ассоциированный со словом *queen* (королева). То есть в векторном пространстве $\boldsymbol{w}_{\text{king}} - \boldsymbol{w}_{\text{man}} + \boldsymbol{w}_{\text{woman}} \approx \boldsymbol{w}_{\text{queen}}$. Аналогичные результаты получаются и для других семантических связей, например: $\boldsymbol{w}_{\text{France}} - \boldsymbol{w}_{\text{Paris}} + \boldsymbol{w}_{\text{London}} \approx \boldsymbol{w}_{\text{England}}$, – и то же самое справедливо для многих других стран и городов.

Это привело к постановке задачи о *разрешении аналогий* (analogy solving), в которой различные погружения слов оцениваются по их способности отвечать на вопросы об аналогии вида *man*:*woman*!*king*:*?* путем решения задачи оптимизации:

$$\text{analogy}(m : w \to k : ?) = \underset{v \in V \setminus \{m,w,k\}}{\text{argmax}} \cos(\boldsymbol{v}, \boldsymbol{k} - \boldsymbol{m} + \boldsymbol{w}). \tag{11.4}$$

В работе Levy and Goldberg [2014] замечено, что для нормированных векторов решение задачи максимизации (11.4) эквивалентно решению уравнения (11.5), т. е. поиску слова, которое похоже на king, похоже на man, но не похоже на woman:

$$\text{analogy}(m : w \to k : ?) = \underset{v \in V \setminus \{m,w,k\}}{\text{argmax}} \cos(\boldsymbol{v}, \boldsymbol{k}) - \cos(\boldsymbol{v}, \boldsymbol{m}) + \cos(\boldsymbol{v}, \boldsymbol{w}). \tag{11.5}$$

Леви и Гольдберг называют этот метод 3CosAdd. Переход от арифметики слов к арифметике сходства слов позволяет в какой-то степени объяснить, почему погружения слов способны «разрешать» аналогии, а также подсказывает, какие именно аналогии можно восстановить таким методом. Кроме того, становится очевиден потенциальный недостаток метода 3CosAdd: из-за аддитивной природы целевой функции один член суммы может доминировать в выражении, что приведет к игнорированию всех прочих. Леви и Гольдберг предложили смягчить эту проблему, заменив аддитивную функцию мультипликативной (3CosMul):

$$\text{analogy}(m : w \to k : ?) = \underset{v \in V \setminus \{m,w,k\}}{\text{argmax}} \frac{\cos(\boldsymbol{v}, \boldsymbol{k}) - \cos(\boldsymbol{v}, \boldsymbol{m})}{\cos(\boldsymbol{v}, \boldsymbol{w}) + \epsilon}. \tag{11.6}$$

Хотя задача разрешения аналогий находит применение для оценки погружений слов, не ясно, какой вывод относительно качества погружений можно сделать из успешности такого теста: пригодны ли они для чего-то, кроме решения этой конкретной задачи?

11.8. Донастройка и проекции

Довольно часто получающееся сходство не полностью отражает то сходство, которое имел в виду разработчик приложения. Бывает так, что исследователь составил сам или получил доступ к репрезентативному и сравнительно большому списку пар слов, отражающему желаемое сходство лучше, чем погружения слов, но обладающему меньшим покрытием. Метод *донастройки* (retrofitting), предложенный в работе Faruqui et al. [2015], позволяет использовать такие данные для повышения качества матрицы погружений слов. Эффективность этого подхода продемонстрирована путем использования информации, полученной из баз данных WordNet и PPDB (раздел 6.2.1), для улучшения предобученных векторов погружений.

Метод предполагает наличие предобученной матрицы погружений $\boldsymbol{E}$ и графа $\mathcal{G}$, который кодирует сходство слов, – вершинами графа являются слова, и два слова считаются похожими, если их вершины соединены ребром. Отметим, что представление в виде графа очень общее, а список пар слов, считающихся похожими, легко укладывается в этот каркас. Принцип работы метода заключается в решении задачи оптимизации, которая ищет новую матрицу погружений слов $\hat{\boldsymbol{E}}$, строки которой близки как к соответствующим строкам $\boldsymbol{E}$, так и к строкам, соответствующим их соседям в графе $\mathcal{G}$. Точнее, целевая функция имеет вид:

$$\underset{\hat{\boldsymbol{E}}}{\operatorname{argmin}} \sum_{i=1}^{n} \left(\alpha_i \left\| \hat{\boldsymbol{E}}_{[w_i]} - \boldsymbol{E}_{[w_i]} \right\|^2 + \sum_{(w_i, w_j) \in \mathcal{G}} \beta_{ij} \left\| \hat{\boldsymbol{E}}_{[w_i]} - \hat{\boldsymbol{E}}_{[w_j]} \right\|^2 \right), \tag{11.7}$$

где α_i и β_{ij} отражают важность сходства слова с самим собой или с другими словами. На практике α_i обычно принимают равными 1, а β_{ij} – числами, обратными степени вершины w_i графа (если у слова много соседей, то оно меньше влияет на каждого из них). Этот подход прекрасно работает на практике.

Родственная проблема возникает, когда имеется две матрицы погружений: одна с маленьким словарем $\boldsymbol{E}^{\mathrm{S}} \in \mathbb{R}^{|V_{\mathrm{S}}| \times d_{\mathrm{emb}}}$, а другая – с большим словарем $\boldsymbol{E}^{\mathrm{L}} \in \mathbb{R}^{|V_{\mathrm{S}}| \times d_{\mathrm{emb}}}$, которые обучались независимо и, стало быть, несовместимы. Быть может, матрица с маленьким словарем обучалась с помощью более дорогостоящего алгоритма (скажем, как часть более крупной и сложной сети), а матрица с большим словарем загружена из веба. Имеет место частичное перекрытие словарей, и мы хотим использовать векторы из большей матрицы $\boldsymbol{E}^{\mathrm{L}}$ для представления слов, отсутствующих в меньшей матрице $\boldsymbol{E}^{\mathrm{S}}$. Заполнить разрыв между двумя пространствами погружения можно с помощью линейной проекции[1] [Kiros et al., 2015, Mikolov et al., 2013]. Цель обучения – найти хорошую матрицу проекции $\boldsymbol{M} \in \mathbb{R}^{d_{\mathrm{emb}} \times d_{\mathrm{emb}}}$, которая

[1] Разумеется, чтобы это работало, необходимо предположить наличие линейного отношения между обоими пространствами. Но на практике метод линейной проекции зачастую дает хорошие результаты.

отображала бы строки $\boldsymbol{E}^{\mathrm{L}}$ так, чтобы они были близки к соответствующим строкам $\boldsymbol{E}^{\mathrm{S}}$, для чего требуется решить следующую задачу оптимизации:

$$\underset{\boldsymbol{M}}{\operatorname{argmin}} \sum_{w \in V_S \cap V_L} \left\| \boldsymbol{E}^{\mathrm{L}}_{[w]} \cdot \boldsymbol{M} - \boldsymbol{E}^{\mathrm{S}}_{[w]} \right\|. \tag{11.8}$$

Затем обученную матрицу можно использовать также для проецирования тех строк $\boldsymbol{E}^{\mathrm{L}}$, для которых нет соответствующих строк в $\boldsymbol{E}^{\mathrm{S}}$. Этот подход был успешно использован в работе Kiros et al. [2015], чтобы увеличить размер словаря для LSTM-кодировщика предложений (модель кодирования предложений, описанная в этой работе, обсуждается в разделе 17.3).

Еще одно любопытное (хотя и не очень надежное) применение метода проецирования описано в работе Mikolov et al. [2013], где обучались матрицы для проецирования векторов погружений, обученных на языке A (скажем, английском), на векторы погружений, обученные на языке B (скажем, испанском), на основе начального списка известных переводов отдельных слов.

11.9. Практические вопросы и подводные камни

Хотя из интернета можно скачать готовые предобученные погружения слов, не рекомендуется слепо использовать такие погружения, считая их черным ящиком. Различные решения, в том числе выбор обучающего корпуса (но необязательно его размер: больше не всегда лучше, а меньший по размеру, но лучше вычищенный или в большей степени ориентированный на предметную область корпус часто оказывается более эффективным в конкретной ситуации), выбор контекстов для определения дистрибутивного сходства и многие гиперпараметры обучения могут оказать заметное влияние на результаты. При наличии аннотированного тестового набора для задачи о вычислении сходства стоит поэкспериментировать с несколькими вариантами настройки и выбрать ту, которая лучше работает на контрольном наборе. Обсуждение различных гиперпараметров и их влияния на сходство *см*. в работе Levy et al. [2015].

При использовании готовых векторов погружений рекомендуется использовать те же схемы лексемизации и нормализации текста, которые применялись при обработке корпуса.

Наконец, сходство, индуцированное векторами слов, основано на дистрибутивных сигналах, поэтому на него распространяются все ограничения дистрибутивных методов установления сходства, описанные в разделе 10.7. О них следует помнить, работая с векторами слов.

Глава 12

Пример: применение архитектуры прямого распространения для вывода смысла предложения

В разделе 11.6 мы ввели сумму попарных сходств слов как ориентир для решения задачи о сходстве коротких документов. Если даны два предложения, первое – составленное из слов $w_1^1, \ldots, w_{\ell_1}^1$, и второе – составленное из слов $w_1^2, \ldots, w_{\ell_2}^2$, то с каждым словом ассоциирован предобученный вектор слов $\boldsymbol{w}_{1:\ell_1}^1$ и $\boldsymbol{w}_{1:\ell_2}^2$ соответственно, а сходство между документами определяется как

$$\sum_{i=1}^{\ell_1}\sum_{j=1}^{\ell_2}\mathrm{sim}\left(\boldsymbol{w}_i^1, \boldsymbol{w}_j^2\right).$$

Это полезный ориентир для сравнения, притом полученный безо всякого участия учителя. Сейчас мы покажем, как оценку сходства документов можно значительно улучшить, если имеется источник обучающих данных. В своем изложении мы будем исходить из сети, предложенной в работе Parikh et al. [2016] для задачи семантического вывода *Stanford Natural Language Inference* (SNLI). Помимо строгой модели для решения этой задачи, в данной работе продемонстрировано также, как основные описанные выше компоненты можно сочетать в разных слоях, получив тем самым сложную и мощную сеть, совместно обученную для данной задачи.

12.1. Естественно-языковой вывод и набор данных NLI

В задаче *естественно-языкового вывода*, называемой также *распознаванием текстуальной импликации* (recognizing textual entailment – RTE), дано два текста s_1 и s_2, и требуется решить, верно ли, что s_1 влечет за собой s_2 (т. е. что s_2 можно вывести из s_1), или что s_1 противоречит s_2, или что тексты нейтральны (ни один не вытекает из другого и ни один не противоречит другому). Примеры предложений, иллюстрирующих различные случаи, приведены в табл. 12.1.

Таблица 12.1. Задача естественно-языкового вывода (текстуальной импликации). Примеры взяты из тестовой части набора данных SNLI

	Двое мужчин на велосипедах участвуют в гонке
Следствие	Люди едут на велосипедах
Нейтральное	Мужчины едут на велосипедах по улице
Противоречие	Несколько человек ловят рыбу
	Два врача оперируют пациента
Следствие	Врачи оперируют
Нейтральное	Два врача оперируют мужчину
Противоречие	Два хирурга обедают

Задача естественно-языкового вывода была впервые изучена в работе Dagan and Glickman [2004], а затем подвергалась ряду тестов, известных под названием PASCAL RTE Challenges [Dagan et al., 2005]. Эта задача очень трудна[1], ее точное решение означало бы понимание текста на уровне человека. Подробное обсуждение этой задачи и подходов к ее решению, не связанных с нейронными сетями, *см*. в книге Дагана, Рота, Сэммонса и Занзотто, вышедшей в этой же серии [Dagan et al., 2013].

SNLI – большой набор данных, созданный в работе Bowman et al. [2015], он содержит 570 000 написанных людьми пар предложений, каждая из которых вручную аннотирована как следствие, противоречие или нейтральная. Для создания предложений людям предъявляли подписи к изображениям и просили, не видя изображения, придумать подпись, которая наверняка являлась бы правильным описанием изображения (следствие), потенциально могла бы быть правильным описанием (нейтральная) и наверняка являлась бы неправильным описанием (противоречие). После того как было собрано 570 000 таких пар, 10 % из них были подвергнуты дополнительному контролю – их предъявляли другим людям и просили классифицировать пару как следствие, нейтральную или противоречие. Затем проверенные пары использовались для формирования тестового и контрольного наборов. Примеры в табл. 12.1 взяты из набора SNLI[2].

Этот набор проще существовавших ранее наборов данных для задачи RTE, но намного больше и по-прежнему далеко не тривиален (особенно когда требуется отличить следствия от нейтральных пар). Набор SNLI часто используется для оценивания моделей вывода смысла. Заметим, что задача отнюдь не сводится к установлению попарного сходства слов; рассмотрим, к примеру, второе предложение в табл. 12.1: нейтральное предложение гораздо больше похоже (в терминах среднего сходства слов) на исходное, чем предложение-следствие. Мы должны уметь повышать значимость одних видов сходства, понижать значимость других и понимать, какие виды сходства сохраняют смысл (например, переход от *муж-*

---

[1] Описанный ниже набор данных SNLI содержит в основном описания сцен, представленных на изображениях, он проще общей задачи RTE без ограничений, для решения которой могут понадобиться весьма сложные шаги вывода. Примером логической пары в неограниченной задаче RTE могут служить предложения *About two weeks before the trial started, I was in Shapiro's office in Century City* ⇒ *Shapiro worked in Century City* (За две недели до начала испытаний я был в офисе Шапиро в Сенчури Сити ⇒ Шапиро работал в Сенчури Сити).

[2] Но переведены на русский язык. – *Прим. перев.*

чины к *пациенту* в контексте хирургической операции), а какие добавляют новую информацию (например, переход от *пациента* к *мужчине*). Архитектура сети спроектирована в расчете на такие рассуждения.

12.2. Сеть для установления сходства текстов

Сеть будет состоять из нескольких стадий. На первой стадии наша цель – вычислить попарные сходства слов, подходящие для решения этой задачи. Функция сходства двух векторов слов определяется следующим образом:

$$\begin{gathered}\text{sim}(\boldsymbol{w}_1, \boldsymbol{w}_2) = \text{MLP}^{\text{transform}}(\boldsymbol{w}_1) \cdot \text{MLP}^{\text{transform}}(\boldsymbol{w}_2), \\ \text{MLP}^{\text{transform}}(\boldsymbol{x}) \in \mathbb{R}^{d_s}, \quad \boldsymbol{w}_1, \boldsymbol{w}_2 \in \mathbb{R}^{d_{\text{emb}}}.\end{gathered} \tag{12.1}$$

Это значит, что мы сначала преобразуем каждый вектор слова с помощью обученного линейного преобразования, а затем берем скалярное произведение преобразованных векторов.

Каждое слово предложения $\boldsymbol{a}$ может быть похоже на несколько слов предложения $\boldsymbol{b}$, и наоборот. Для каждого слова w_i^a предложения $\boldsymbol{a}$ мы вычисляем ℓ_b-мерный вектор его сходств со словами предложения $\boldsymbol{b}$, нормированный посредством softmax, чтобы все сходства были положительны, а их сумма равнялась 1. Результат называется *вектором выравнивания* для слова:

$$\boldsymbol{\alpha}_i^a = \text{softmax}(\text{sim}(\boldsymbol{w}_i^a, \boldsymbol{w}_1^b), \ldots, \text{sim}(\boldsymbol{w}_i^a, \boldsymbol{w}_{\ell_b}^b)). \tag{12.2}$$

Аналогично вычисляются векторы выравнивания для всех слов $\boldsymbol{b}$:

$$\begin{gathered}\boldsymbol{\alpha}_i^b = \text{softmax}(\text{sim}(\boldsymbol{w}_1^a, \boldsymbol{w}_i^b), \ldots, \text{sim}(\boldsymbol{w}_{\ell_a}^a, \boldsymbol{w}_i^b)). \\ \boldsymbol{\alpha}_i^a = \mathbb{N}_+^{\ell_b}, \quad \boldsymbol{\alpha}_i^b = \mathbb{N}_+^{\ell_a}.\end{gathered}$$

Для каждого слова $\boldsymbol{w}_i^a$ мы вычисляем вектор $\bar{\boldsymbol{w}}_i^b$, состоящий из взвешенной суммы слов в $\boldsymbol{b}$, которые выровнены с $\boldsymbol{w}_i^a$, и аналогично для каждого слова $\boldsymbol{w}_j^b$:

$$\begin{aligned}\bar{\boldsymbol{w}}_i^b &= \sum_{j=1}^{\ell_b} \boldsymbol{\alpha}_{i[j]}^a \boldsymbol{w}_j^b; \\ \bar{\boldsymbol{w}}_j^a &= \sum_{i=1}^{\ell_a} \boldsymbol{\alpha}_{i[j]}^b \boldsymbol{w}_i^a.\end{aligned} \tag{12.3}$$

Вектор $\bar{\boldsymbol{w}}_i^b$ улавливает взвешенную смесь слов предложения $\boldsymbol{b}$, которые затрагивает i-е слово предложения $\boldsymbol{a}$.

Такие представления последовательности векторов в виде взвешенных сумм, в которых веса вычисляются функцией softmax от оценок, как в уравнении (12.2), часто называются *механизмом внимания*. Название связано с тем, что веса отражают важность каждого объекта целевой последовательности для заданного исходного объекта – сколько *внимания* следует уделить каждому объекту целевой последовательности по отношению к исходному объекту. Более подробно мы обсудим механизм внимания в главе 17, когда будем рассматривать модели условного порождения.

Сходство между $\boldsymbol{w}_i^a$ и соответствующей смесью $\bar{\boldsymbol{w}}_i^b$ в предложении $\boldsymbol{b}$ необязательно релевантно задаче естественно-языкового вывода. Мы пытаемся преобразовать каждую такую пару в вектор представления $\boldsymbol{v}_i^a$, который фокусируется на информации, важной для задачи. Это делает еще одна сеть прямого распространения:

$$\begin{aligned}\boldsymbol{v}_i^a &= \text{MLP}^{\text{pair}}([\boldsymbol{w}_i^a; \bar{\boldsymbol{w}}_i^b]);\\ \boldsymbol{v}_j^b &= \text{MLP}^{\text{pair}}([\boldsymbol{w}_j^b; \bar{\boldsymbol{w}}_j^a]).\end{aligned} \tag{12.4}$$

Заметим, что, в отличие от функции сходства (12.1), в которой каждый член рассматривается индивидуально, эта функция может обрабатывать оба члена поразному.

Наконец, мы складываем получившиеся векторы и передаем их МСП-классификатору для предсказания отношения между двумя предложениями (следствие, противоречие или нейтральное):

$$\begin{aligned}\boldsymbol{v}^a &= \sum_i \boldsymbol{v}_i^a;\\ \boldsymbol{v}^b &= \sum_j \boldsymbol{v}_j^b;\\ \hat{\boldsymbol{y}} &= \text{MLP}^{\text{decide}}([\boldsymbol{v}^a; \boldsymbol{v}^b]).\end{aligned} \tag{12.5}$$

В работе Parikh et al. [2016] все МСП имеют два скрытых уровня размера 200 и функции активации ReLU. Весь процесс описывается одним графом вычислений, и производится сквозное обучение сети с перекрестной энтропией в качестве потери. Сами предобученные погружения слов *не* изменялись вместе с остальной сетью, полагаясь на то, что МСП сумеет выполнить преобразование, необходимое для адаптации. На момент написания книги эта архитектура давала наилучшие результаты на наборе данных SNLI.

Подведем итоги. *Преобразующая* сеть обучает функцию сходства для выравнивания на уровне слов. Она преобразует каждое слово в нечто сохраняющее важные сходства на уровне слов. По завершении работы этой сети каждый вектор слова похож на векторы других слов, которые с большой вероятностью относятся к той же сущности или к тому же событию. Цель этой сети – найти слова, которые *могли бы* внести вклад в логический вывод. Мы получаем выравнивания в обоих направлениях: каждое слово $\boldsymbol{a}$ с несколькими словами $\boldsymbol{b}$ и каждое слово $\boldsymbol{b}$ с несколькими словами $\boldsymbol{a}$. Выравнивания гибкие и выражаются в форме членства во взвешенной группе, а не в виде однозначных решений, так что слово может участвовать в нескольких похожих парах. Эта сеть, скорее всего, поместит рядом пары слов *men* и *people*, *men* и *two* и *man* и *patient*, а также флективные формы *perform* и *performing*.

Затем сеть *оценки пар* анализирует каждую выровненную пару (слово + группа), применяя взвешенное CBOW-представление, и выделяет информацию, релевантную паре. Полезна ли эта пара для предсказания логического следования? Сеть также рассматривает предложение, из которого взят каждый элемент пары, и, вероятно, обучится тому, что из слова *patient* может следовать *man*, но не наоборот.

Наконец, *решающая* сеть анализирует агрегированные данные, полученные из пар слов, и принимает решение на основе всех имеющихся фактов. Рассуждение

распадается на три стадии: сначала выявляются слабые локальные факты в терминах выравнивания по сходству, затем анализируются взвешенные комбинации нескольких слов и добавляется направленность, после чего на третьей стадии все локальные факты объединяются в глобальное решение.

Гиперпараметры сети настроены под эту конкретную задачу и набор данных, и не ясно, будет ли сеть обобщаться на другие конфигурации. В этой главе мы хотели не столько познакомить читателя с конкретной архитектурой сети, сколько продемонстрировать, что спроектировать сложную архитектуру возможно, и иногда для этого стоит приложить усилия. Из нового материала, описанного в этой главе, отметим использование весов *мягкого выравнивания* $\boldsymbol{\alpha}_i^a$ (этот механизм называют также *вниманием*) для вычисления взвешенной суммы элементов $\bar{\boldsymbol{w}}_i^b$ (уравнение (12.3)). Мы еще встретимся с этой идеей в главе 17, когда будем обсуждать основанное на внимании условное порождение с помощью рекуррентных нейронных сетей.

Часть III

СПЕЦИАЛИЗИРОВАННЫЕ АРХИТЕКТУРЫ

В предыдущих главах мы обсудили обучение с учителем и нейронные сети прямого распространения и их применение к лингвистическим задачам. Нейронные сети прямого распространения применяются в основном в архитектурах классификации общего назначения – ничто в них не «заточено» специально под лингвистические данные или последовательности. И действительно, по большей части мы структурировали лингвистические задачи так, чтобы они соответствовали каркасу МСП.

В следующих главах мы будем изучать некоторые нейросетевые архитектуры, в большей степени специализированные для работы с лингвистическими данными. В частности, мы обсудим одномерные сверточно-пулинговые архитектуры и рекуррентные нейронные сети (РНС). Сверточные нейронные сети (СНС) предназначены для выявления информативных *n*-грамм и *n*-грамм с пропусками в текстовых последовательностях независимо от их позиции, но с учетом локальных паттернов упорядочения. РНС проектировались для улавливания неочевидных паттернов и закономерностей в последовательностях, они позволяют моделировать немарковские зависимости, поскольку анализируют «бесконечные окна» вокруг фокусного слова, обращая особое внимание на информативные последовательные паттерны. Наконец, мы обсудим модели генерации последовательностей и условной генерации.

ВЫДЕЛЕНИЕ ПРИЗНАКОВ Архитектуры СНС и РНС, рассматриваемые в этой части книги, используются преимущественно как *экстракторы признаков*. Сеть типа СНС и РНС не является автономным компонентом, она порождает векторы (или последовательности векторов), которые затем подаются на вход другим частям сети, а уже те в конечном итоге выдают предсказания. Сеть подвергается сквозному обучению (часть, занимающаяся предсказанием, и сверточная, или рекуррентная, часть обучаются совместно), так что векторы, порождаемые СНС/РНС, улавливают аспекты входных данных, полезные для конкретной задачи предсказания. В следующих главах мы познакомимся с экстракторами признаков на основе архитектур СНС и РНС. На момент написания книги экстракторы на базе РНС чаще применялись в приложениях для работы с текстом, чем экстракторы на базе СНС. Однако у разных архитектур разные плюсы и минусы, так что в будущем баланс может сместиться в другую сторону. Полезно знать обе архитек-

туры, к тому же приобретают популярность гибридные подходы. В главах 16 и 17 обсуждается интеграция экстракторов признаков на основе РНС с различными архитектурами предсказания и порождения в NLP. Многое из сказанного будет относиться и к сверточным сетям.

СНС И РНС КАК ДЕТАЛИ КОНСТРУКТОРА Изучая архитектуры СНС и РНС, полезно рассматривать их как «детали конструктора», которые можно по-разному соединять для создания требуемой структуры и достижения желаемого поведения.

Такой подход поощряется механизмом графа вычислений и градиентной оптимизации. Он позволяет трактовать различные архитектуры – МСП, СНС, РНС – как компоненты или блоки, из которых строятся все более и более крупные конструкции, – нужно только следить за тем, чтобы размерности входа и выхода соединяемых компонент совпадали, а обо всем остальном позаботятся граф вычислений и обучение градиентными методами.

Это позволяет создавать большие и сложные сетевые структуры с несколькими слоями МСП, СНС и РНС, передающими данные между собой, и производить сквозное обучение такой сети. В последующих главах рассмотрено несколько примеров, но на самом деле их гораздо больше – для разных задач подходят разные архитектуры. Изучая новую архитектуру, подходите к ней не с вопросом «какие существующие компоненты она замещает?» или «как использовать ее для решения моей задачи?», а с вопросом «как включить ее в мой арсенал строительных блоков и объединить с другими компонентами для достижения желаемого результата?».

Глава 13

Детекторы *n*-грамм: сверточные нейронные сети

Иногда нас интересуют предсказания, сделанные на основе упорядоченных наборов элементов (последовательностей слов в предложении, последовательностей предложений в документе и т. д.). Рассмотрим, к примеру, предсказания эмоциональной окраски (положительной, отрицательной или нейтральной) таких предложений:

- *Part of the charm of Satin Rouge is that it avoids the obvious with humor and lightness* (Своим очарованием «Сатин Руж» отчасти обязан тому, что с юмором и легкостью избегает очевидного);
- *Still, this flick is fun and host to some truly excellent sequences* (Вместе с тем это радостный фильм, в котором есть поистине великолепные сцены).

Некоторые слова в предложениях четко указывают на эмоциональную окраску (*charm*, *fun*, *excellent*), другие менее информативны (*Still*, *host*, *flick*, *lightness*, *obvious*, *avoids*), и в хорошем приближении можно сказать, что информативность слова не зависит от его позиции в предложении. Мы хотели бы подать обучаемому всю последовательность слов и дать возможность процессу обучения выявить важные ключи. Одно из возможных решений – передать представление в виде CBOW полносвязной сети типа МСП. Но у CBOW есть недостаток – полностью игнорируется информация о порядке слов, т. е. представления предложений «*it was not good, it was actually quite bad*» (это было не хорошо, на самом деле это было очень плохо) и «*it was not bad, it was actually quite good*» (это было не плохо, на самом деле это было очень хорошо) в точности совпадают. Для задачи классификации глобальные позиции индикаторов «*not good*» и «*not bad*» не имеют значения, однако локальный порядок слов (тот факт, что слово «*not*» встречается непосредственно перед словом «*bad*») очень важен. Аналогично в предложении *Montias pumps a lot of energy into his nuanced narative, and surrounds himself with a cast of quirky – but not stereotyped – street characters* (Монтиас наполняет украшенное тонкими нюансами повествование огромной энергией и окружает себя странноватыми – но не стереотипными – уличными персонажами) имеется существенное различие между «not stereotyped» (положительный индикатор) и «not nuanced» (без всяких нюансов) (отрицательный индикатор). Приведенные выше примеры – это простые случаи отрицания, но встречаются и менее очевидные фразы, например «avoids the obvious» (избегает очевидного) и «avoids the charm» (избегает очарования) в первом примере. Короче говоря, анализ *n*-грамм гораздо информативнее анализа мешка слов.

При наивном подходе можно было бы попробовать погружать пары слов (биграммы) или тройки слов (триграммы), а не отдельные слова, и строить CBOW погруженных *n*-грамм. Такая архитектура действительно весьма эффективна, но приводит к гигантским матрицам погружений, плохо масштабируется на более широкие *n*-граммы и страдает от проблем разреженности данных, поскольку статистическая сила разных *n*-грамм не обобществляется (погружения биграмм «quite good» и «very good» совершенно независимы, поэтому, увидев одну из них в процессе обучения, обучаемый не сможет сделать никакого вывода о другой, исходя только из составляющих слов).

АРХИТЕКТУРА CHC В этой главе мы познакомимся со сверточно-пулинговой архитектурой (или сверточными нейронными сетями, CHC), которая хорошо приспособлена к этой проблеме моделирования. Сверточная нейронная сеть предназначена для выявления характерных локальных предикторов в большой структуре и их объединения в представление этой структуры вектором фиксированного размера; при этом она улавливает те локальные аспекты, которые наиболее информативны для решаемой задачи предсказания. То есть сверточная архитектура выявит *n*-граммы, обладающие предсказательной силой для задачи, не заставляя нас заранее задавать вектор погружения для каждой возможной *n*-граммы. (В разделе 13.2 мы обсудим альтернативный метод, который позволяет работать с неограниченным словарем *n*-грамм, так что размер матрицы погружения остается ограниченным.) Сверточная архитектура позволяет также обобществлять предсказательную силу *n*-грамм с общими компонентами, даже если точно такая же *n*-грамма не встречалась на этапе обучения.

Сверточную архитектуру можно было бы развернуть в иерархию сверточных слоев, так что каждый последующий анализирует все более длинные *n*-граммы в предложении. Она также допускает построение модели, чувствительной к несмежным *n*-граммам. Этот вопрос обсуждается в разделе 13.3.

Как было отмечено во введении к этой части, CHC, по существу, является архитектурой для выделения признаков. Она предназначена не для использования в качестве автономной полезной сети, а для включения в объемлющую сеть, вместе с которой и обучается. На слой CHC возлагается ответственность за выделение осмысленных подструктур, полезных для общей задачи предсказания.

ИСТОРИЯ И ТЕРМИНОЛОГИЯ Сверточно-пулинговые архитектуры [LeCun and Bengio, 1995] развивались в сообществе нейронных сетей для компьютерного зрения, где продемонстрировали огромный успех в качестве детекторов объектов – распознавание объекта из заранее заданной категории («кошка», «велосипед») независимо от его положения в изображении [Krizhevsky et al., 2012]. В применении к изображениям в архитектуре использовались свертки на двумерной сетке. В применении к текстам нас в основном интересуют одномерные свертки (предложений). Сверточные сети были представлены сообществу NLP в пионерской работе Collobert et al. [2011], где использовались для пометки семантических ролей, а впоследствии Калхбреннер (Kalchbrenner et al. [2014]) и Ким (Kim [2014]) применили их для классификации по эмоциональной окраске и по типу вопроса.

Поскольку своими корнями сверточные архитектуры уходят в компьютерное зрение, оттуда и из теории обработки сигналов вообще заимствованы многие термины: *фильтр*, *канал*, *рецептивное поле*. Мы расскажем, что они означают, когда будем вводить соответствующие понятия.

13.1. Свертка + пулинг – основы

Основная идея применения сверточно-пулинговой архитектуры к лингвистическим задачам – вызвать нелинейную (обученную) функцию для каждого вхождения скользящего окна длиной k слов в предложение[1]. Эта функция (которая еще называется «фильтром») преобразует окно в скалярное значение. Можно применить несколько таких фильтров, получив в результате ℓ-мерный вектор (по одному элементу для каждого фильтра), улавливающий важные свойства слов в окне. Затем используется операция «пулинга», которая объединяет векторы, построенные для разных окон, в один ℓ-мерный вектор, для чего берет максимальное или среднее значение элементов, встретившихся в ℓ-мерных векторах для разных окон. Идея в том, чтобы сконцентрироваться на самых важных «признаках» предложения независимо от их позиции, – каждый фильтр выделяет из окна свой индикатор, а операция пулинга фокусируется на важных индикаторах. Получившиеся ℓ-мерные векторы подаются на вход сети, которая использует их для предсказания. Градиенты функции потерь, вычисленные в процессе обратного распространения на этапе обучения, используются для настройки параметров фильтра, чтобы выделить аспекты данных, наиболее важные для решаемой задачи. Интуитивно понятно, что когда скользящее окно длины k пробегает по последовательности, фильтр обучается выявлять информативные k-граммы. На рис. 13.2 иллюстрируется применение сверточно-пулинговой сети к последовательности.

13.1.1. Одномерная свертка текста

Начнем с операции одномерной свертки[2]. А в следующем разделе перейдем к пулингу.

Рассмотрим последовательность слов $w_{1:n} = w_1, \ldots, w_n$, для каждого из которых определено соответствующее ему d_{emb}-мерное погружение слова $\boldsymbol{E}_{[w_i]} = \boldsymbol{w_i}$. Одномерная свертка ширины k вычисляется путем сдвига скользящего окна размера k вдоль предложения и применения одного и того же «фильтра» к каждому окну в последовательности, где под фильтром понимается скалярное произведение с весовым вектором $\boldsymbol{u}$, за которым часто следует нелинейная функция активации. Определим оператор $\oplus(\boldsymbol{w_{i:i+k-1}})$ как конкатенацию векторов $\boldsymbol{w_i}, \ldots, \boldsymbol{w_{i+k-1}}$. Тогда конкатенированный вектор i-го окна равен $\boldsymbol{x_i} = [\boldsymbol{w_i}; \boldsymbol{w_{i+1}}; \ldots; \boldsymbol{w_{i+k-1}}]$, $\boldsymbol{x_i} \in \mathrm{R}^{k \times d_{\text{emb}}}$.

Затем мы применяем фильтр к каждому вектору окна, что дает скалярные значения p_i:

$$p_i = g(\boldsymbol{x_i} \cdot \boldsymbol{u}); \tag{13.1}$$

$$\boldsymbol{x_i} = \oplus(\boldsymbol{w_{i:i+k-1}}), \tag{13.2}$$

$$p_i \in \mathbb{R}, \quad \boldsymbol{x_i} \in \mathbb{R}^{k \cdot d_{\text{emb}}}, \quad \boldsymbol{u} \in \mathbb{R}^{k \cdot d_{\text{emb}}},$$

где g – нелинейная функция активации.

Принято использовать ℓ разных фильтров $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_\ell$, которые можно собрать в матрицу $\boldsymbol{U}$, и еще часто добавляют вектор смещения $\boldsymbol{b}$:

---

[1] Размер окна k иногда называют *рецептивным полем* свертки.

[2] Мы говорим «одномерная», имея в виду, что свертка применяется к одномерным выходным данным – последовательностям, а не к двумерным – изображениям.

$$\boldsymbol{p}_i = g(\boldsymbol{x}_i \cdot \boldsymbol{U} + \boldsymbol{b}), \qquad (13.3)$$
$$\boldsymbol{p}_i \in \mathbb{R}^{\ell}, \quad \boldsymbol{x}_i \in \mathbb{R}^{k \cdot d_{\text{emb}}}, \quad \boldsymbol{U} \in \mathbb{R}^{k \cdot d_{\text{emb}} \times \ell}, \quad \boldsymbol{b} \in \mathbb{R}^{\ell}.$$

Каждый вектор $\boldsymbol{p}_i$ – это набор значений, представляющих (или обобщающих) i-е окно. В идеале каждое измерение улавливает самостоятельный вид индикативной информации.

УЗКИЕ И ШИРОКИЕ СВЕРТКИ Сколько у нас векторов $\boldsymbol{p}_i$? В предложении длины n при окне размера k существует $n - k + 1$ позиций начала последовательности, так что получается $n - k + 1$ векторов $\boldsymbol{p}_{1:n-k+1}$. Это называется *узкой сверткой*. Альтернатива – дополнить предложение $k - 1$ словами с каждой стороны, тогда получится $n + k + 1$ векторов $\boldsymbol{p}_{1:n+k+1}$. Это называется *широкой сверткой* [Kalchbrenner et al., 2014]. Количество получающихся векторов будем обозначать буквой m.

АЛЬТЕРНАТИВНАЯ ФОРМУЛИРОВКА СВЕРТКИ В нашем описании сверток по последовательности n элементов $w_{1:n}$ с каждым элементом ассоциирован d-мерный вектор, и все эти векторы конкатенируются в большой $1 \times d \cdot n$ вектор предложения. Тогда сверточная сеть с окном размера k и ℓ выходными значениями основана на матрице размера $k \cdot d \times \ell$. Эта матрица применяется к сегментам вектора предложения размера $1 \times d \cdot n$, которые соответствуют окнам длины k. Каждое такое умножение дает ℓ значений, каждое из которых можно рассматривать как результат скалярного произведения вектора длины $k \cdot d + 1$ (строки матрицы) и сегмента вектора предложения.

Еще одна (эквивалентная) формулировка, часто встречающаяся в литературе, подразумевает, что n векторов составлены в вертикальную стопку, так что получается матрица предложения размера $n \times d$. Тогда операция свертки выполняется путем перемещения различных матриц $k \times d$ (которые называются «ядрами» или «фильтрами») по матрице предложения и выполнения *матричной свертки* между каждым ядром и соответствующим сегментом матрицы предложения. Операция матричной свертки двух матриц определяется как результат их поэлементного умножения и суммирования результатов. Каждая операция свертки с ядром предложения порождает одно значение, так что всего получается ℓ значений. Легко убедиться, что оба подхода действительно эквивалентны; надо только заметить, что каждое ядро соответствует строке матрицы размера $k \cdot d \times \ell$, а свертка с ядром соответствует скалярному произведению на строку матрицы.

На рис. 13.1 показана узкая и широкая свертки в обеих интерпретациях.

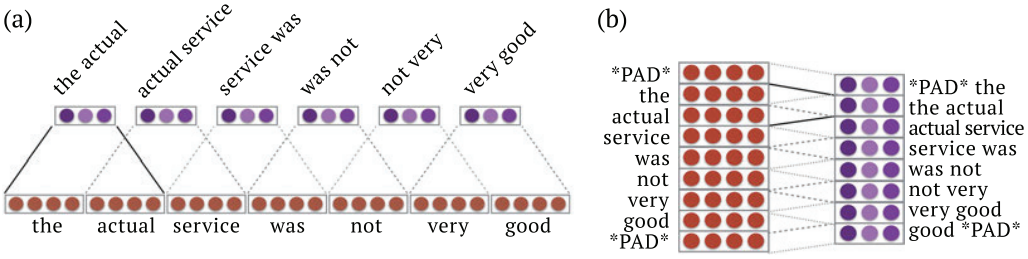


Рис. 13.1 ❖ Входы и выходы узкой и широкой сверток в двух интерпретациях: конкатенации векторов и стопки векторов. (a) Узкая свертка с окном размера $k = 2$ и трехмерным выходом ($\ell = 3$), когда используется конкатенация векторов. (b) Широкая свертка с окном размера $k = 2$ и трехмерным выходом ($\ell = 3$), когда используется стопка векторов

КАНАЛЫ В компьютерном зрении изображение представляется набором пикселей, каждый из которых описывает яркость цвета в одной точке. Если используется цветовая схема RGB, то каждый пиксель – это комбинация трех значений яркости, по одной для красной, зеленой и синей компонент. Эти компоненты хранятся в трех разных матрицах. Каждая матрица дает свой «взгляд» на изображение и называется *каналом*. В компьютерном зрении свертка изображения обычно означает применение разных наборов фильтров к каждому каналу с последующим объединением трех получившихся векторов в один. Приняв метафору «разных взглядов на изображение», мы можем определить также разные каналы при обработке текста. Например, одним каналом будет последовательность слов, а другим – последовательность соответствующих меток частей речи. Применение свертки к словам дает векторы $\boldsymbol{p}^w_{1:m}$, а применение к меткам частей речи – векторы $\boldsymbol{p}^t_{1:m}$. Оба взгляда можно затем объединить, просуммировав $\boldsymbol{p}_i = \boldsymbol{p}^w_i + \boldsymbol{p}^t_i$ или произведя конкатенацию $\boldsymbol{p}_i = [\boldsymbol{p}^w_i; \boldsymbol{p}^t_i]$.

ИТОГИ Основная идея сверточного слоя – применить одну и ту же параметрическую функцию ко всем k-граммам последовательности. При этом создается последовательность m векторов, каждый из которых представляет одну k-грамму. Это представление чувствительно к самим словам, составляющим k-грамму, и к порядку их следования внутри k-граммы, но от положения k-граммы в последовательности оно не зависит.

13.1.2. Пулинг векторов

Применение свертки к тексту дает m векторов $\boldsymbol{p}_{1:m}$, где $\boldsymbol{p}_i \in \mathbb{R}^\ell$. Эти векторы затем объединяются (подвергаются *пулингу*) в один вектор $\boldsymbol{c} \in \mathbb{R}^\ell$, представляющий все предложение. В идеале вектор $\boldsymbol{c}$ будет улавливать существенно важную информацию в последовательности. Природа информации, которая должна быть закодирована в векторе $\boldsymbol{c}$, зависит от задачи. Например, если мы выполняем классификацию по эмоциональной окраске, то важны информативные n-граммы, обозначающие эмоции, а если тематическую классификацию, то информативные n-граммы, которые указывают на определенную тему.

В процессе обучения вектор $\boldsymbol{c}$ подается в последующие слои сети (т. е. в МСП), и в итоге – в выходной слой, который выдает предсказания[1]. Процедура обучения сети вычисляет потерю на задаче предсказания, а градиенты ошибки распространяются обратно до слоев свертки и пулинга, а также до слоев погружения. Процесс обучения настраивает матрицу свертки $\boldsymbol{U}$, вектор смещения $\boldsymbol{b}$, следующую далее сеть и потенциально также матрицу погружений $\boldsymbol{E}$ так, чтобы вектор $\boldsymbol{c}$, получающийся в результате операций свертки и пулинга, действительно кодировал информацию, релевантную задаче[2].

---

[1] Входом в последующие слои сети может быть как сам вектор $\boldsymbol{c}$, так и комбинация $\boldsymbol{c}$ с другими векторами.

[2] Помимо пользы для предсказаний, у процедуры обучения есть и побочный эффект: установка параметров $\boldsymbol{W}$, $\boldsymbol{B}$ и погружений $\boldsymbol{E}$, которые можно использовать в сверточно-пулинговой архитектуре для кодирования предложений произвольной длины векторами фиксированного размера таким образом, что предложения, обладающие общей прогностической информацией, будут близки друг к другу.

MAX-ПУЛИНГ Из всех операций пулинга наибольшее распространение получил *max-пулинг*, при котором берется максимальный элемент по каждому измерению:

$$\boldsymbol{c}_{[j]} = \max_{1 \le i \le m} \boldsymbol{p}_{i[j]}, \quad \forall j \in [1, \ell], \tag{13.4}$$

где $\boldsymbol{p}_{i[j]}$ – j-й элемент $\boldsymbol{p}_i$. Смысл операции max-пулинга состоит в том, чтобы получить самую характерную информацию среди всех позиций окна. В идеале каждое измерение должно «специализироваться» на определенном виде предикторов, а операция max выбирает самый важный предиктор каждого вида.

На рис. 13.2 иллюстрируется процесс свертки и max-пулинга.

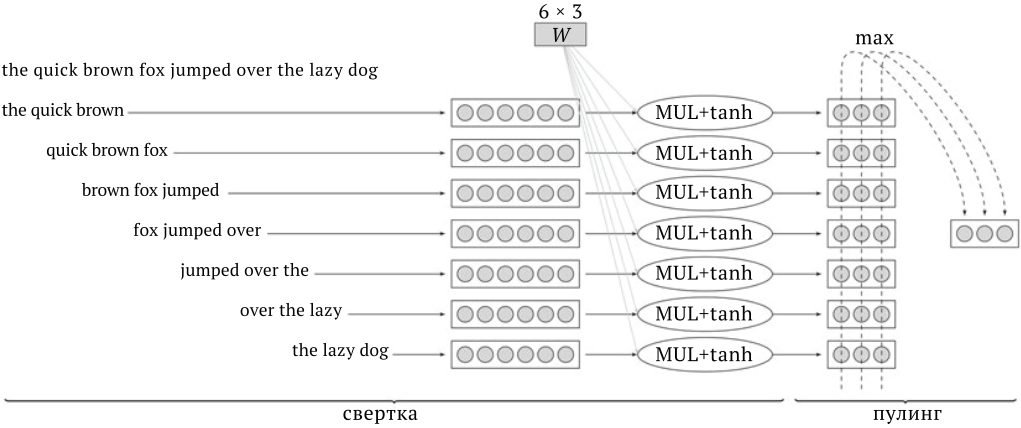


Рис. 13.2 ❖ Одномерная свертка + пулинг для последовательности «the quick brown fox jumped over the lazy dog». Это узкая свертка (предложение не дополняется фиктивными словами) с окном размера 3. Каждое слово представляется двумерным вектором погружения (не показан). Затем векторы погружения конкатенируются, что дает шестимерные представления окон. Каждое из семи окон пропускается через фильтр 6×3 (линейное преобразование, за которым следует поэлементное применение tanh), и в результате получается семь тримерных профильтрованных представлений. Затем применяется операция max-пулинга, которая выбирает максимум по каждому измерению, что дает окончательный тримерный пулинговый вектор

ПУЛИНГ УСРЕДНЕНИЕМ Следующим по популярности типом пулинга является пулинг усреднением, т. е. усреднение по каждому измерению вместо взятия максимального элемента:

$$\boldsymbol{c} = \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{p}_i. \tag{13.5}$$

Одна из интерпретаций пулинга усреднением – образование непрерывного мешка (CBOW) k-граммных представлений, получающихся в результате свертки, а не из слов предложения.

K-MAX-ПУЛИНГ Еще один вариант, предложенный в работе Kalchbrenner et al. [2014], – операция k-max-пулинга, при которой для каждого измерения оставля-

ются первые k значений, а не только наилучшее, причем сохраняется порядок их появления в тексте[1]. Например, рассмотрим следующую матрицу:

$$\begin{bmatrix} 1 & 2 & 3 \\ 9 & 6 & 5 \\ 2 & 3 & 1 \\ 7 & 8 & 1 \\ 3 & 4 & 1 \end{bmatrix}.$$

1-max-пулинг по векторам-столбцам даст вектор [9 8 5], тогда как 2-max-пулинг – матрицу $\begin{bmatrix} 9 & 6 & 3 \\ 7 & 8 & 5 \end{bmatrix}$, строки которой затем конкатенируются, так что получается вектор [9 6 3 7 8 5].

Операция k-max-пулинга позволяет выделить k самых активных индикаторов, которые могут находиться в несмежных позициях; она сохраняет порядок признаков, но нечувствительна к их конкретным позициям. Она также может более точно распознать, сколько раз признак давал высокую активацию [Kalchbrenner et al., 2014].

ДИНАМИЧЕСКИЙ ПУЛИНГ Вместо того чтобы выполнять одну операцию пулинга по всей последовательности, мы иногда хотим сохранять некоторую позиционную информацию, исходя из понимания предметной области. Для этого мы можем разбить векторы $\boldsymbol{p}_i$ на r групп, применить пулинг к каждой группе по отдельности, а затем конкатенировать r получившихся ℓ-мерных векторов $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_r$. Разбиение векторов $\boldsymbol{p}_i$ на группы производится на основе знаний о предметной области. Например, мы можем решить, что слова в начале предложения более индикативны, чем встречающиеся позже. Затем можно разбить последовательность на r участков равного размера и применить к каждому операцию max-пулинга. Например, в работе Johnson and Zhang [2015] показано, что для тематической классификации документов полезно применить пулинг усреднением к 20 участкам, четко отделяющим начальные предложения (из которых обычно становится ясна тема) от последующих, тогда как в задаче классификации по эмоциональной окраске оптимальна одна операция max-пулинга по всему предложению (в предположении, что для определения эмоций достаточно одного или двух очень сильных сигналов независимо от места в предложении).

Аналогично в задаче извлечения отношений нам предлагают два слова и просят определить отношение между ними. Мы могли бы высказать гипотезу, что слова перед первым словом, слова после второго слова и слова между ними дают три разных вида информации [Chen et al., 2015]. Поэтому мы соответственно разбиваем векторы $\boldsymbol{p}_i$ на группы и производим пулинг отдельно для окон, принадлежащих каждой группе.

---

[1] В этой главе буквой k обозначается размер окна свертки. В k-max-пулинге k имеет совершенно другой смысл. Мы решили оставить букву k, чтобы не порывать с принятыми традициями.

13.1.3. Вариации

Вместо одного сверточного слоя можно параллельно применить несколько таких слоев. Например, можно завести четыре сверточных слоя, каждый со своим размером окна от 2 до 5, которые улавливают последовательности *k*-грамм разной длины. Результат каждого сверточного слоя затем подвергается пулингу, получившиеся векторы конкатенируются и подаются на вход следующему слою [Kim, 2014].

Сверточная архитектура не ограничена линейным упорядочением предложения. Так, в работе Ma et al. [2015] операция свертки обобщена на деревья синтаксических зависимостей. Каждое окно окружает вершина синтаксического дерева, а пулинг производится по разным вершинам. Аналогично в работе Liu et al. [2015] сверточная архитектура применена к путям зависимостей, выделенным из деревьев зависимостей. А в работе Le and Zuidema [2015] предложено применять max-пулинг к векторам, представляющим различные выводы, ведущие к одному и тому же элементу схемы в анализаторе схем.

13.2. Альтернатива: хеширование признаков

Сверточные сети для обработки текста работают как очень эффективные детекторы признаков для последовательных *k*-грамм. Однако они требуют большого количества операций умножения матриц, так что объемом вычислений нельзя пренебречь. С точки зрения быстродействия, эффективнее было бы использовать погружения *k*-грамм напрямую, а затем подвергать *k*-граммы пулингу усреднением (получая в результате представления в виде непрерывного мешка *n*-грамм) или max-пулингу. Недостаток такого подхода в том, что он требует выделения отдельного вектора погружения для каждой возможной *k*-граммы, что может оказаться неприемлемым с точки зрения объема памяти, поскольку количество *k*-грамм в обучающем корпусе может быть очень велико.

Решением проблемы является использование техники *хеширования признаков*, которая берет начало в линейных моделях [Ganchev and Dredze, 2008, Shi et al., 2009, Weinberger et al., 2009], а недавно была адаптирована к нейронным сетям [Joulin et al., 2016]. Идея заключается в том, что мы не вычисляем заранее отображение словаря в индекс. Вместо этого мы выделяем память для матрицы погружений $\boldsymbol{E}$ с N строками (N должно быть достаточно, но не запредельно большим, скажем, порядка миллионов или десятков миллионов). Встретив на этапе обучения *k*-грамму, мы назначаем ей строку $\boldsymbol{E}$, применив *хеш-функцию* h, которая детерминированно отображает ее в число в диапазоне $[1, N]$, $i = h(k\text{-грамма}) \in [1, N]$. Затем мы используем соответствующую строку $\boldsymbol{E}_{[h(k\text{-грамма})]}$ как вектор погружения. Таким образом, каждой *k*-грамме динамически присваивается индекс без необходимости явно хранить соответствие между *k*-граммами и индексами или выделять вектор погружения для каждой *k*-граммы. Из-за конфликтов хеширования некоторым *k*-граммам может быть сопоставлен один и тот же вектор погружения (действительно, поскольку пространство возможных *k*-грамм намного больше количества выделенных векторов погружения, такие конфликты неизбежны), но, поскольку большинство *k*-грамм неинформативно для решаемой задачи, конфликты сглаживаются процессом обучения. Если требуется большая тщательность, то

можно использовать несколько разных хеш-функций $h_1, \ldots, h_r$ и представлять каждую k-грамму суммой строк, соответствующих ее хешам ($\sum_{i=1}^{r} \boldsymbol{E}_{[h_i(k\text{-грамма})]}$. Тогда если информативная k-грамма конфликтует с другой информативной k-граммой по одному хешу, то с большой вероятностью она не будет конфликтовать по какому-то из остальных хешей.

Этот прием с хешированием (его еще называют *хеш-ядром*) очень хорошо работает на практике и дает чрезвычайно эффективные модели на основе мешка n-грамм. Рекомендуется начинать именно с него и только потом переходить к более сложным подходам или архитектурам, используя полученные результаты для сравнения.

13.3. Иерархические свертки

Описанный выше одномерный подход можно рассматривать как детектор n-грамм. Сверточный слой с окном размера k обучается выявлять индикативные k-граммы во входных данных.

Этот метод можно обобщить на *иерархию сверточных слоев*, когда несколько сверточных слоев применяются один за другим. Обозначим $\text{CONV}^k_\Theta(\boldsymbol{w}_{1:n})$ результат применения свертки с окном размера k и параметрами Θ к каждому окну размера k в последовательности $\boldsymbol{w}_{1:n}$:

$$\begin{aligned} \boldsymbol{p}_{1:m} &= \text{CONV}^k_{U,b}(\boldsymbol{w}_{1:n}); \\ \boldsymbol{p}_i &= g(\oplus(\boldsymbol{w}_{i:i+k-1}) \cdot \boldsymbol{U} + \boldsymbol{b}); \\ m &= \begin{cases} n-k+1 & \text{узкая свертка} \\ n+k+1 & \text{широкая свертка} \end{cases}. \end{aligned} \tag{13.6}$$

Теперь можно определить последовательность r сверточных слоев, каждый из которых передает данные следующему за ним:

$$\begin{aligned} \boldsymbol{p}^1_{1:m_1} &= \text{CONV}^{k_1}_{U^1,b^1}(\boldsymbol{w}_{1:n}); \\ \boldsymbol{p}^2_{1:m_2} &= \text{CONV}^{k_2}_{U^2,b^2}(\boldsymbol{p}^1_{1:m_1}); \\ &\ldots \\ \boldsymbol{p}^r_{1:m_r} &= \text{CONV}^{k_r}_{U^r,b^r}(\boldsymbol{p}^{r-1}_{1:m_{r-1}}). \end{aligned} \tag{13.7}$$

Получающиеся векторы $\boldsymbol{p}^r_{1:m_r}$ улавливают все более широкие окна (*рецептивные поля*) последовательности. Если имеется r слоев с окном размера k, то каждый вектор $\boldsymbol{p}^r_i$ будет чувствителен к окну, содержащему $r(k-1)+1$ слов[1]. Более того, вектор $\boldsymbol{p}^r_i$ чувствителен к *n-граммам с пропусками*, содержащим $k + r - 1$ слов, т. е. потенциально может улавливать такие словосочетания, как «*not ___ good*» или «*obvious*

[1] Чтобы понять, почему это так, примем во внимание, что первый сверточный слой преобразует каждую последовательность k соседних векторов слов в векторы, представляющие k-граммы. Затем второй сверточный слой объединяет k векторов последовательных k-грамм в векторы, улавливающие окна длиной $k + (k-1)$ слов, и т. д. Таким образом, r-я свертка будет улавливать $k + (r-1)(k-1)$ слов.

___ predictable ___ plot», где ___ – короткая последовательность слов, а также более сложные паттерны, в которых структура пропусков уточнена (например, «последовательность слов, не содержащих *not*» или «последовательность слов, похожих на наречия»)[1]. На рис. 13.3 показана двухуровневая иерархическая свертка с $k = 2$.

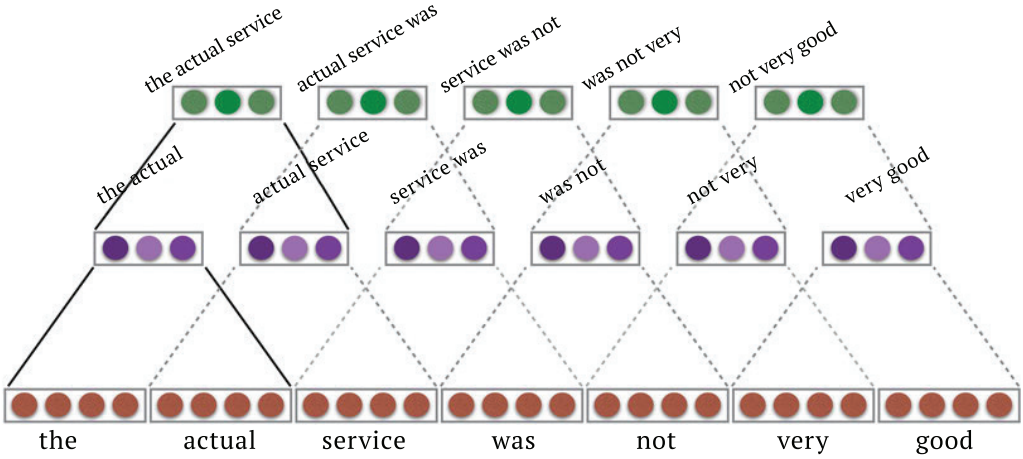


Рис. 13.3 ❖ Двухуровневая иерархическая свертка с $k = 2$

ШАГИ, ДЫРЫ И ПУЛИН До сих пор мы применяли операцию свертки к каждому окну последовательности длиной k слов, т. е. к окнам, начинающимся в позициях 1, 2, 3, ... Говорят, что такая свертка имеет *шаг* (stride) 1. Возможны и большие шаги, например свертка с шагом 2 применяется к окнам, начинающимся в позициях 1, 3, 5, ... Вообще, определим $\text{CONV}^{k,s}$ как:

$$\begin{aligned} \boldsymbol{p}_{1:m} &= \text{CONV}^{k,s}_{\boldsymbol{U},\boldsymbol{b}}(\boldsymbol{w}_{1:n}); \\ \boldsymbol{p}_i &= g(\oplus(\boldsymbol{w}_{1+(i-1)s:(s+k)i}) \cdot \boldsymbol{U} + \boldsymbol{b}), \end{aligned} \tag{13.8}$$

где s – величина шага. В результате на выходе сверточного слоя получится более короткая последовательность.

В *дырявой сверточной архитектуре* (dilated convolution architecture) [Strubell et al., 2017, Yu and Koltun, 2016] применяется иерархия сверточных слоев с шагом $k - 1$ каждый (т. е. $\text{CONV}^{k,k-1}$). Тем самым допускается экспоненциальный рост эффективного размера окна как функции от количества слоев. На рис. 13.4 показаны сверточные слои с разной величиной шага, а на рис. 13.5 – дырявая сверточная архитектура.

У дырявой архитектуры есть альтернатива – оставить фиксированный шаг 1, но уменьшать длину последовательности при переходе от слоя к слою, применяя

[1] Чтобы убедиться в этом, рассмотрим применение последовательности двух сверточных слоев с размером окна 2 каждый к последовательности слов *funny and appealing*. Первый слой кодирует биграммы *funny and* и *and appealing* векторами и может решить, что в результирующих векторах следует оставить эквиваленты «*funny* ___» и «___ *appealing*». Второй слой затем может объединить их в «*funny* ___*appealing*», «*funny* ___» или «___*appealing*».

локальный пулинг, т. е. соседние k'-граммы векторов можно преобразовать в один вектор с помощью max-пулинга или пулинга усреднением. Даже если подвергать пулингу только каждые два соседних вектора, то сверточно-пулинговый слой в иерархии уменьшит длину последовательности вдвое. Как и в случае дырявой архитектуры, мы получаем экспоненциальное убывание длины последовательности как функции количества слоев.

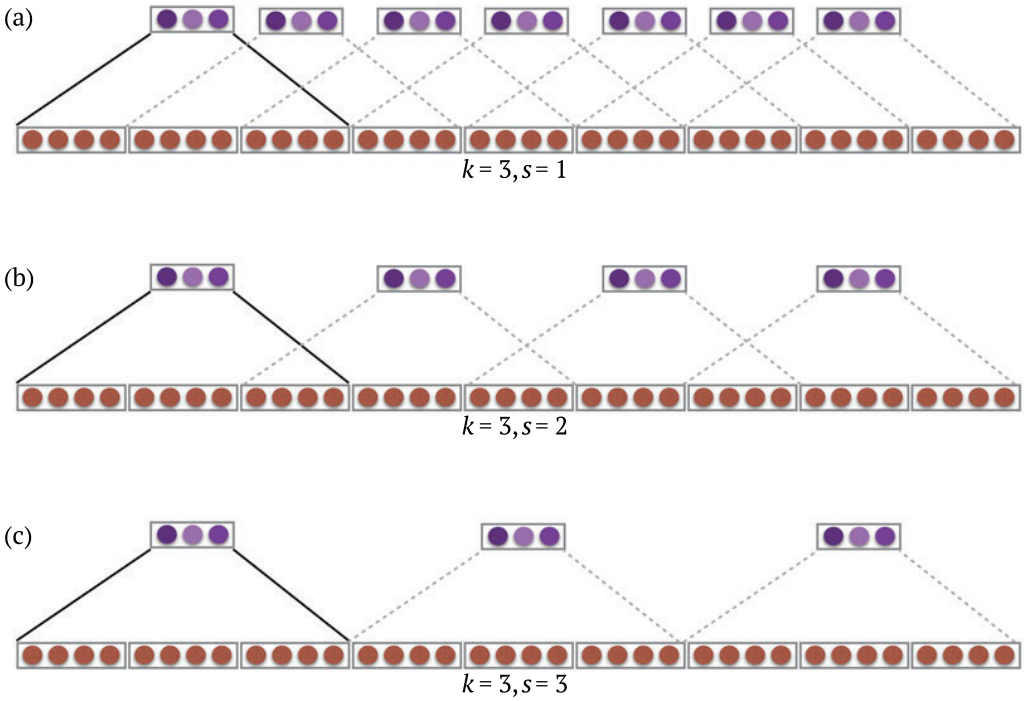


Рис. 13.4 ❖ Шаги: (a–c) сверточный слой с $k = 3$ и шагами 1, 2, 3

СВЯЗЫВАНИЕ ПАРАМЕТРОВ И ПРЯМЫЕ СВЯЗИ Еще одна вариация, применимая к сверточной архитектуре, – *связывание параметров* (parameter-tying), когда во всех параметрических слоях используется один и тот же набор параметров $\boldsymbol{U}, \boldsymbol{b}$. В результате достигается лучшее разделение параметров, а кроме того, мы получаем возможность использовать неограниченное число сверточных слоев (поскольку все сверточные слои разделяют одни и те же параметры, то количество слоев необязательно задавать заранее), что, в свою очередь, позволяет сводить к одному вектору последовательности произвольной длины, применяя цепочку узких сверток, каждая из которых уменьшает длину последовательности векторов.

Иногда при работе с глубокими архитектурами бывают полезны *прямые связи* (skip-connection): они подают на вход i-му слою не только выходные векторы $(i - 1)$-го слоя, но и векторы с предыдущих слоев, которые объединяются с векторами $(i - 1)$-го слоя с помощью конкатенации, усреднения или суммирования.

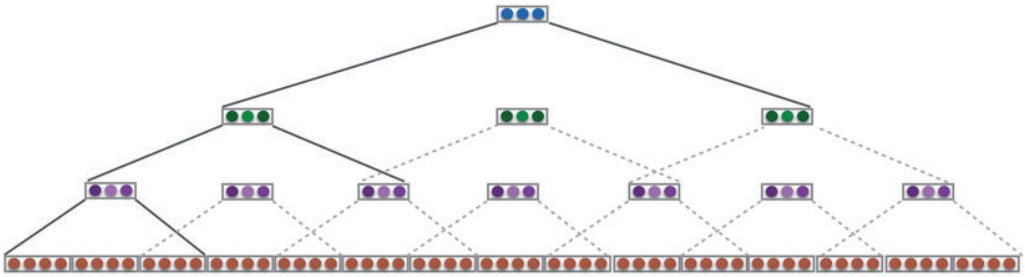


Рис. 13.5 ❖ Трехуровневая дырявая иерархическая свертка с $k = 3$

Для дальнейшего чтения. Иерархические и дырявые сверточно-пулинговые архитектуры широко распространены среди специалистов по компьютерному зрению, которые предложили разнообразные глубокие архитектуры, содержащие много сверточных и пулинговых слоев с различными шагами, что позволило получить очень сильные результаты в области классификации изображений и распознавания объектов [He et al., 2016, Krizhevsky et al., 2012, Simonyan and Zisserman, 2015]. Применение таких глубоких архитектур в NLP пока еще находится в зачаточной стадии. В работе Zhang et al. [2015] описаны первые эксперименты по классификации текстов на основе иерархической свертки литер, а в работе Conneau et al. [2016] эти результаты получили развитие на сей раз с очень глубокими сверточными сетями. В работе Strubell et al. [2017] имеется хороший обзор иерархических и дырявых архитектур для задачи разметки последовательности. В работе Kalchbrenner et al. [2016] дырявые свертки выступают в роли кодировщиков в архитектуре кодировщик–декодер (раздел 17.2) для машинного перевода. В работе Xiao and Cho [2016] иерархия сверток с локальным пулингом применена к последовательности литер в задаче классификации документов, а затем результирующие векторы загружены в рекуррентную нейронную сеть. Мы вернемся к этому примеру в разделе 16.2.2, после того как обсудим рекуррентные нейронные сети.

Глава 14

Рекуррентные нейронные сети: последовательности и стеки

При работе с лингвистическими данными часто приходится иметь дело с последовательностями, например словами (последовательности букв), предложениями (последовательности слов) и документами. Мы видели, что сети прямого распространения могут приспособиться к произвольным функциям выделения признаков из последовательностей благодаря конкатенации и сложению векторов (непрерывный мешок слов – CBOW). В частности, представления в виде CBOW позволяют кодировать последовательности произвольной длины векторами фиксированного размера. Однако CBOW-представление крайне ограничено и вынуждает игнорировать порядок признаков. Сверточные сети также позволяют кодировать последовательность вектором фиксированного размера. Хотя представления на основе сверточных сетей – шаг вперед по сравнению с CBOW, поскольку чувствительны к порядку слов, но эта чувствительность ограничена по преимуществу локальными паттернами, а порядок паттернов, расположенных далеко друг от друга, не учитывается[1].

Рекуррентные нейронные сети (РНС) [Elman, 1990] позволяют представить последовательные входные данные произвольного размера векторами фиксированного размера, не отбрасывая структурных свойств данных. РНС, особенно с вентильными архитектурами, например LSTM и GRU, – чрезвычайно эффективное средство улавливания статистических закономерностей в последовательных данных. Пожалуй, это самый весомый вклад глубокого обучения в инструментарий статистической обработки естественного языка.

В этой главе РНС описываются с абстрактной точки зрения: как интерфейс для преобразования входной последовательности в выход фиксированного размера,

[1] Однако, как было отмечено в разделе 13.3, у иерархических и дырявых сверточных архитектур имеется потенциальная возможность улавливать сравнительно дальние зависимости внутри последовательности.

который можно использовать как компонент объемлющей сети. Обсуждаются различные архитектуры, в которых РНС выступает в роли компонента. В следующей главе мы поговорим о конкретных реализациях абстракции РНС и опишем РНС Элмана (ее еще называют простой РНС), долгую краткосрочную память (Long-Short-Term Memory – LSTM) и вентильный рекуррентный блок (Gated Recurrent Unit – GRU). Затем в главе 16 мы рассмотрим примеры моделирования проблем NLP с помощью РНС.

В главе 9 мы обсуждали языковое моделирование и марковское предположение. РНС допускают языковые модели, не удовлетворяющие марковскому предположению, в которых следующее слово обусловлено всей историей (т. е. всеми предшествующими словами предложения). Это открывает путь *моделям условной генерации*, когда языковая модель, используемая как генератор, обусловлена каким-то другим сигналом, например предложением на другом языке. Более подробно такие модели описываются в главе 17.

14.1. Абстракция РНС

Будем обозначать $\boldsymbol{x}_{i:j}$ последовательность векторов $\boldsymbol{x}_i, \ldots, \boldsymbol{x}_j$. На верхнем уровне РНС – это функция, которая принимает упорядоченную последовательность n d_{in}-мерных векторов произвольной длины $\boldsymbol{x}_{1:n} = \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ ($\boldsymbol{x}_i \in \mathbb{R}^{d_{in}}$) и возвращает один d_{out}-мерный вектор $\boldsymbol{y}_n \in \mathbb{R}^{d_{out}}$:

$$\begin{gathered} \boldsymbol{y}_n = \mathrm{RNN}(\boldsymbol{x}_{1:n}). \\ \boldsymbol{x}_i \in \mathbb{R}^{d_{in}}, \quad \boldsymbol{y}_n \in \mathbb{R}^{d_{out}}. \end{gathered} \tag{14.1}$$

Тем самым неявно определяется выходной вектор $\boldsymbol{y}_i$ для каждого префикса $\boldsymbol{x}_{1:i}$ последовательности $\boldsymbol{x}_{1:n}$. Будем обозначать RNN^* функцию, возвращающую эту последовательность:

$$\begin{gathered} \boldsymbol{y}_{1:n} = \mathrm{RNN}^*(\boldsymbol{x}_{1:n}); \\ \boldsymbol{y}_i = \mathrm{RNN}(\boldsymbol{x}_{1:i}); \\ \boldsymbol{x}_i \in \mathbb{R}^{d_{in}}, \quad \boldsymbol{y}_i \in \mathbb{R}^{d_{out}}. \end{gathered} \tag{14.2}$$

Затем выходной вектор $\boldsymbol{y}_n$ используется для предсказания. Например, модель для предсказания условной вероятности события e при условии последовательности $\boldsymbol{x}_{1:n}$ можно определить как $p(e = j|\boldsymbol{x}_{1:n}) = \mathrm{softmax}(\mathrm{RNN}(\boldsymbol{x}_{1:n}) \cdot \boldsymbol{W} + \boldsymbol{b})_{[j]}$, т. е. j-й элемент выходного вектора, получающегося применением softmax к линейному преобразованию результата РНС-кодирования $\boldsymbol{y}_n = \mathrm{RNN}(\boldsymbol{x}_{1:n})$. Функция RNN определяет механизм обусловливания всей историей $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i$, не требуя выполнения описанного в главе 9 марковского предположения, которое традиционно использовалось при моделировании последовательностей. На самом деле языковые модели на основе РНС дают очень хорошие оценки перплексивности по сравнению с моделями, основанными на n-граммах.

На более детальном уровне РНС определяется рекурсивно с помощью функции R, которая принимает вектор состояния $\boldsymbol{s}_{i-1}$ и входной вектор $\boldsymbol{x}_i$, а возвращает новый вектор состояния $\boldsymbol{s}_i$. Затем вектор состояния $\boldsymbol{s}_i$ отображается в выходной

вектор $\boldsymbol{y}_i$ с помощью простой детерминированной функции $O(\cdot)$[1]. Базой рекурсии является начальный вектор состояния $\boldsymbol{s}_0$, который служит также входом в РНС. Для краткости мы часто будем опускать начальный вектор $\boldsymbol{s}_0$ или предполагать, что он нулевой.

При построении РНС, как и при построении сети прямого распространения, необходимо задать размерность входных данных $\boldsymbol{x}_i$, а также размерность выхода $\boldsymbol{y}_i$. Размерность состояния $\boldsymbol{s}_i$ является функцией от размерности выхода[2]:

$$\begin{aligned} \mathrm{RNN}^*(\boldsymbol{x}_{1:n}; \boldsymbol{s}_0) &= \boldsymbol{y}_{1:n}; \\ \boldsymbol{y}_i &= O(\boldsymbol{s}_i); \\ \boldsymbol{s}_i &= R(\boldsymbol{s}_{i-1}, \boldsymbol{x}_i); \\ \boldsymbol{x}_i \in \mathbb{R}^{d_{in}}, \quad \boldsymbol{y}_i \in \mathbb{R}^{d_{out}}, \quad \boldsymbol{s}_i &\in \mathbb{R}^{f(d_{out})}. \end{aligned} \tag{14.3}$$

Функции R и O одинаковы для всех позиций последовательности, но РНС отслеживает состояние вычисления с помощью вектора состояния $\boldsymbol{s}_i$, который сохраняется и передается между вызовами R.

Графически РНС традиционно представляется, как показано на рис. 14.1.

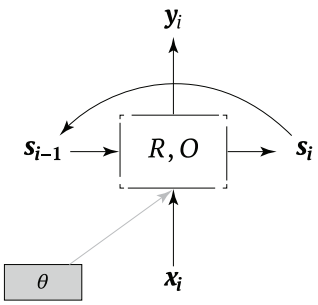


Рис. 14.1 ❖ Графическое представление РНС (рекурсивное)

Это представление повторяет рекурсивное определение и корректно для произвольно длинных последовательностей. Однако для входной последовательности конечного размера (а все рассматриваемые нами последовательности конечны) можно *раскрутить* рекурсию и получить структуру, показанную на рис. 14.2.

Мы включили параметры θ, которые обычно не показываются на графических иллюстрациях, чтобы подчеркнуть тот факт, что на всех временных шагах используются одни и те же параметры. При разных R и O получаются разные сетевые структуры, обладающие разными свойствами в терминах времени работы и способности к эффективному обучению градиентными методами. Однако все они со-

---

[1] Нестандартная функция O введена для того, чтобы унифицировать различные РНС-модели, которые будут представлены в следующей главе. Для простой РНС (РНС Элмана) и архитектуры GRU функция O – тождественное отображение, а для архитектуры LSTM функция O выбирает фиксированное подмножество состояния.

[2] Хотя возможны архитектуры РНС, в которых размерность состояния не зависит от размерности выхода, в современных популярных архитектурах, в том числе простой РНС, LSTM и GRU, такая гибкость не предоставляется.

гласованы с одним и тем же абстрактным интерфейсом. Детали конкретных реализаций R и O – простой РНС, LSTM и GRU – мы приведем в главе 15. Но прежде рассмотрим работу с РНС на абстрактном уровне.

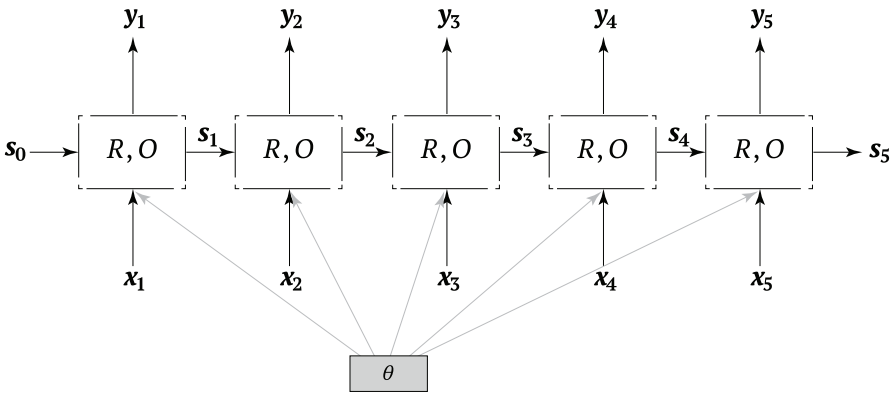


Рис. 14.2 ❖ Графическое представление РНС (после раскрутки рекурсии)

Сначала заметим, что значение $\boldsymbol{s}_i$ (значит, и $\boldsymbol{y}_i$) определяется всеми векторами $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i$. Например, раскрыв рекурсию для $i = 4$, получаем:

$$\begin{aligned}\boldsymbol{s}_4 &= R(\boldsymbol{s}_3, \boldsymbol{x}_4)\\ &= R(\overbrace{R(\boldsymbol{s}_2, \boldsymbol{x}_3)}^{\boldsymbol{s}_3}, \boldsymbol{x}_4)\\ &= R(R(\overbrace{R(\boldsymbol{s}_1, \boldsymbol{x}_2)}^{\boldsymbol{s}_2}, \boldsymbol{x}_3), \boldsymbol{x}_4)\\ &= R(R(R(\overbrace{R(\boldsymbol{s}_0, \boldsymbol{x}_1)}^{\boldsymbol{s}_1}, \boldsymbol{x}_2), \boldsymbol{x}_3), \boldsymbol{x}_4).\end{aligned} \tag{14.4}$$

Таким образом, $\boldsymbol{s}_n$ и $\boldsymbol{y}_n$ можно интерпретировать как *кодирование* всей входной последовательности[1]. Полезно ли такое кодирование? Зависит от того, что понимать под полезностью. Цель обучения сети заключается в том, чтобы установить такие параметры R и O, при которых состояние несет полезную для решаемой задачи информацию.

14.2. Обучение РНС

Глядя на рис. 14.2, легко увидеть, что раскрученная РНС – это просто очень глубокая нейронная сеть (а точнее, очень большой *граф вычислений* с довольно сложными узлами), в которой одни и те же параметры разделяются между многими частями вычисления и в разные слои добавляются дополнительные входные дан-

[1] Заметим, что если при проектировании R не применялись контрмеры, вполне вероятно, что более поздние элементы входной последовательности будут сильнее влиять на $\boldsymbol{s}_n$, чем более ранние.

ные. Поэтому, чтобы обучить РНС, нам всего лишь нужно создать раскрученный граф вычислений для заданной входной последовательности, добавить в него узел потери, а затем использовать алгоритм обратного распространения для вычисления градиентов этой потери. В литературе по РНС эта процедура называется *обратным распространением во времени* (backpropagation through time – BPTT) [Werbos, 1990][1].

Какова целевая функция обучения? Важно понимать, что РНС сама по себе почти ничего не делает, а служит лишь обучаемым компонентом более крупной сети. Окончательное предсказание и вычисление потери производит эта объемлющая сеть, а ошибка обратно распространяется через РНС. Таким образом, РНС обучается кодировать свойства входной последовательности, полезные для последующей задачи предсказания. Сигнал от учителя не применяется к РНС непосредственно, а проходит через объемлющую сеть.

Ниже приведены некоторые типичные архитектуры включения РНС в более крупную сеть.

14.3. Типичные примеры использования РНС

14.3.1. Приемщик

Один из вариантов – класть в основу сигнала от учителя только конечный выходной вектор $\boldsymbol{y}_n$. При таком подходе РНС обучается как *приемщик*. Мы наблюдаем конечное состояние и принимаем решение о результате[2]. Например, рассмотрим обучение РНС, которая читает литеры слова по одной и затем использует конечное состояние, чтобы предсказать, какой частью речи это слово является (*см.* работу Ling et al. [2015b]), РНС, которая читает предложение и, исходя из конечного состояния, решает, несет ли оно положительную или отрицательную эмоциональную окраску (Wang et al. [2015b]), или РНС, которая читает последовательность слов и решает, составляет ли она допустимую именную группу. В таких случаях потеря определяется в терминах функции от $\boldsymbol{y}_n = O(\boldsymbol{s}_n)$. Как правило, выходной вектор РНС $\boldsymbol{y}_n$ подается на вход полносвязного слоя или МСА, который и порождает предсказание. Затем градиенты ошибки обратно распространяются через

[1] Среди вариантов BPTT можно отметить раскрутку РНС только для фиксированного числа входных символов за один раз: сначала раскрутить для входов $\boldsymbol{x}_{1:k}$, получив $\boldsymbol{s}_{1:k}$. Вычислить потери, произвести обратное распространение ошибки по сети (на k шагов назад). Затем раскрутить входы $\boldsymbol{x}_{k+1:2k}$, используя $\boldsymbol{s}_k$ как начальное состояние, и снова распространить ошибку на k шагов назад и т. д. В основе этой стратегии лежит наблюдение, согласно которому для простой РНС после k шагов (при больших k) градиенты имеют тенденцию исчезать, поэтому их можно спокойно опустить. Такая процедура позволяет обучать произвольно длинные последовательности. Для таких вариантов РНС, как LSTM и GRU, которые специально проектировались, чтобы смягчить проблему исчезающего градиента, подобную раскрутку на фиксированное число шагов обосновать сложнее, тем не менее она все еще используется, например для языкового моделирования книги без разбиения на предложения. Похожий вариант раскручивает сеть для всей последовательности на прямом проходе, но из каждой позиции производит обратное распространение градиентов только на k шагов назад.

[2] Терминология заимствована из Finite-State Acceptor – конечного автомата-приемщика. Однако количество состояний РНС потенциально бесконечно, поэтому для отображения состояний на решения нужна какая-то функция, а не таблица соответствия.

остаток последовательности (*см.* рис. 14.3).[1] Потеря может принимать различные формы: перекрестная энтропия, кусочно-линейная функция, зазор и т. д.

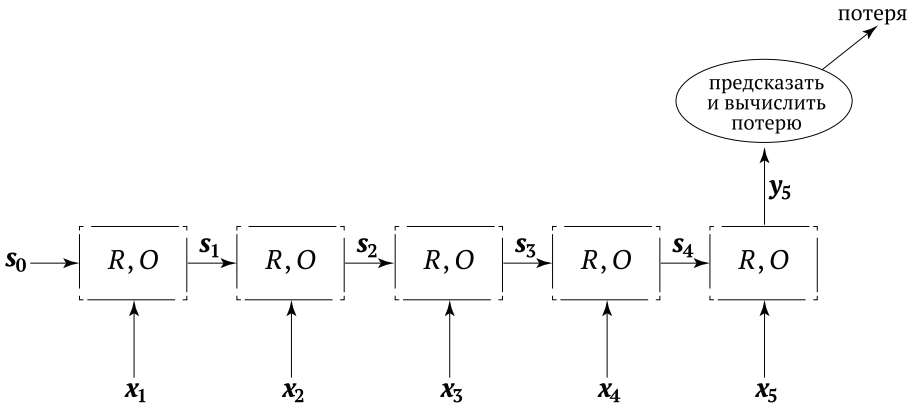


Рис. 14.3 ❖ Граф обучения РНС-приемщика

14.3.2. Кодировщик

Как и в случае приемщика, для обучения кодировщика используется только окончательный выходной вектор $\boldsymbol{y}_n$. Однако, в отличие от приемщика, когда предсказание осуществляется только и исключительно на основе конечного вектора, здесь конечный вектор рассматривается как результат кодирования всей информации в последовательности и используется как дополнительная информация наряду с прочими сигналами. Например, система экстрактивного реферирования документа может сначала прогнать документ через РНС, получив вектор $\boldsymbol{y}_n$, содержащий сводку документа. А затем $\boldsymbol{y}_n$ вместе с другими признаками используется, чтобы выбрать предложения для включения в реферат.

14.3.3. Преобразователь

Еще одна возможность – трактовать РНС как преобразователь (transducer), который порождает выход $\hat{\boldsymbol{t}}_i$ для каждого прочитанного входа. В этом случае мы можем вычислить локальную потерю $L_{\text{local}}(\hat{\boldsymbol{t}}_i, \boldsymbol{t}_i)$ для каждого из выходов $\hat{\boldsymbol{t}}_i$, зная истинную метку $\boldsymbol{t}_i$. Тогда потеря для раскрученной последовательности будет равна $L(\hat{\boldsymbol{t}}_{1:n}, \boldsymbol{t}_{1:n}) = \sum_{i=1}^{n} L_{\text{local}}(\hat{\boldsymbol{t}}_i, \boldsymbol{t}_i)$; можно также использовать вместо суммы другой агрегатор, например среднее или взвешенное среднее (см. рис. 14.4). Один из примеров такого преобразователя – разметчик последовательности, когда мы считаем, что $\boldsymbol{x}_{i:n}$ – представления признаков для n слов последовательности, а $\boldsymbol{t}_i$ – вход для предсказания метки слова i на основе слов 1:i. Основанный на такой архитектуре суперразметчик с комбинаторными категориальными грамматиками (CCG) дает очень сильные результаты [Xu et al., 2015], хотя во многих случаях преобразова-

[1] При таком виде сигнала от учителя обучение на длинных последовательностях может оказаться затруднительным, особенно в случае простой РНС, из-за проблемы исчезающих градиентов. Кроме того, это и вообще трудная задача обучения, поскольку мы ничего не говорим процессу о том, на какие части входных данных обратить особое внимание. И тем не менее во многих случаях работает – и даже очень хорошо.

тель на базе двунаправленной РНС (*см*. раздел 14.4 ниже) лучше подходит для решения таких задач разметки.

Очень естественный пример применения преобразователя дает языковое моделирование, когда последовательность слов $\boldsymbol{x}_{1:i}$ используется для предсказания распределения $(i+1)$-го слова. Показано, что перплексивность языковых моделей на базе РНС гораздо лучше, чем у традиционных [Jozefowicz et al., 2016, Mikolov, 2012, Mikolov et al., 2010, Sundermeyer et al., 2012].

Использование РНС в качестве преобразователей позволяет ослабить марковское предположение, которое по традиции включается в языковые модели и разметчики на основе скрытых марковских моделей, считая условием для предсказания всю предшествующую историю.

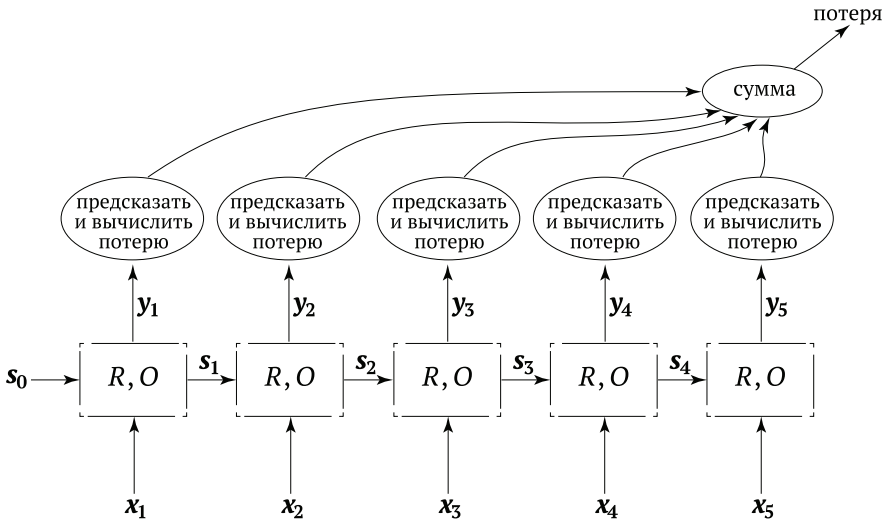


Рис. 14.4 ❖ Граф обучения РНС-преобразователя

Частными случаями РНС-преобразователя являются *РНС-генератор* и связанные с ним архитектуры *условной генерации* (*кодировщик–декодер*) и *условной генерации с вниманием*. Мы будем обсуждать их в главе 17.

14.4. Двунаправленные РНС (biRNN)

Полезное развитие идеи РНС – *двунаправленная РНС* (или просто biRNN) [Graves, 2008, Schuster and Paliwal, 1997][1]. Рассмотрим задачу о разметке последовательности $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$. РНС позволяет вычислить функцию от i-го слова $\boldsymbol{x}_i$ на основе прошлого – слов $\boldsymbol{x}_{1:i}$ вплоть до текущего. Однако для предсказания могут быть полезны и *последующие* слова $\boldsymbol{x}_{i+1:n}$, что с очевидностью вытекает из стандартного подхода со скользящим окном, когда фокусное слово классифицируется на основе k слов, входящих в окружающее его окно. Как РНС ослабляет марковское предположение и позволяет заглядывать сколь угодно далеко в прошлое, так biRNN ослабля-

[1] При использовании в сочетании с конкретной архитектурой РНС, например LSTM, модель называют biLSTM.

ет предположение о фиксированном размере окна, позволяя заглядывать сколь угодно далеко в прошлое и в будущее.

Рассмотрим входную последовательность $\boldsymbol{x}_{1:n}$. В процессе работы biRNN запоминает два различных состояния, $\boldsymbol{s}_i^f$ и $\boldsymbol{s}_i^b$, для каждой входной позиции i. *Прямое состояние* $\boldsymbol{s}_i^f$ основано на $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_i$, а обратное состояние $\boldsymbol{s}_i^b$ – на $\boldsymbol{x}_n, \boldsymbol{x}_{n\text{-}1}, \ldots, \boldsymbol{x}_i$. Прямое и обратное состояния генерируются двумя разными РНС. Первой РНС (R^f, O^f) на вход подается сама последовательность $\boldsymbol{x}_{1:n}$, а второй (R^b, O^b) – инвертированная последовательность. Затем из прямого и обратного состояний строится представление состояния $\boldsymbol{s}_i$. Выход в позиции i основан на конкатенации двух выходных векторов $\boldsymbol{y}_i = [\boldsymbol{y}_i^f; \boldsymbol{y}_i^b] = [O^f(\boldsymbol{s}_i^f); O^b(\boldsymbol{s}_i^b)]$ с учетом прошлого и будущего. Иными словами, $\boldsymbol{y}_i$ – результат кодирования biRNN-сетью i-го слова последовательности – является конкатенацией результатов двух РНС, одна из которых читает последовательность с начала, а другая – с конца.

biRNN$(\boldsymbol{x}_{1:n}, i)$ определяется как выходной вектор, соответствующий i-й позиции последовательности[1]:

$$\text{biRNN}(\boldsymbol{x}_{1:n}, i) = \boldsymbol{y}_i = [\text{RNN}^f(\boldsymbol{x}_{1:i}); \text{RNN}^b(\boldsymbol{x}_{n:i})]. \tag{14.6}$$

Вектор $\boldsymbol{y}_i$ можно затем использовать непосредственно для предсказания или подать на вход более сложной сети. Хотя две РНС работают независимо друг от друга, градиенты ошибки в позиции i распространяются вперед и назад по обеим сетям. Пропуск вектора $\boldsymbol{y}_i$ через МСП до предсказания еще больше смешивает прямые и обратные сигналы. Архитектура biRNN показана на рис. 14.5.

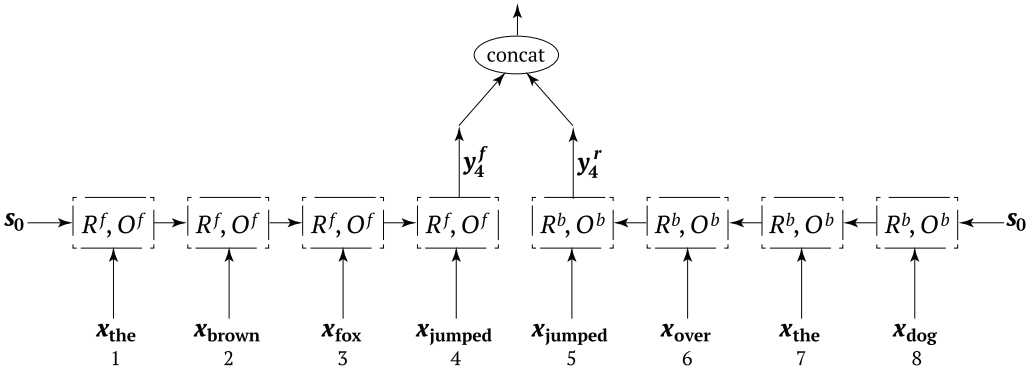


Рис. 14.5 ❖ Вычисление biRNN-представления слова *jumped* в предложении «the brown fox jumped over the dog»

Заметим, что вектор $\boldsymbol{y}_4$, соответствующий слову *jumped*, кодирует бесконечное окно, окружающее (и включающее) фокусный вектор $\boldsymbol{x}_{\text{jumped}}$.

---

[1] Вектор biRNN может быть как простой конкатенацией двух векторов РНС, как в равенстве (14.6), так и конкатенацией, за которой следует линейное преобразование для понижения размерности, часто до размерности входа одной РНС:

$$\text{biRNN}(\boldsymbol{x}_{1:n}, i) = \boldsymbol{y}_i = [\text{RNN}^f(\boldsymbol{x}_{1:i}); \text{RNN}^b(\boldsymbol{x}_{n:i})]\boldsymbol{W}. \tag{14.5}$$

Этот вариант нередко используется при составлении стека из нескольких biRNN, как обсуждается в разделе 14.5.

По аналогии с РНС определим $\text{biRNN}^*(\boldsymbol{x}_{1:n})$ как последовательность векторов $\boldsymbol{y}_{1:n}$:

$$\text{biRNN}^*(\boldsymbol{x}_{1:n}) = \boldsymbol{y}_{i:n} = \text{biRNN}(\boldsymbol{x}_{1:n}, 1), \ldots, \text{biRNN}(\boldsymbol{x}_{1:n}, n). \tag{14.7}$$

Все n выходных векторов $\boldsymbol{y}_i$ можно эффективно вычислить за линейное время, сначала прогнав прямую и обратную РНС, а затем конкатенировав релевантные выходы. Эта архитектура показана на рис. 14.6.

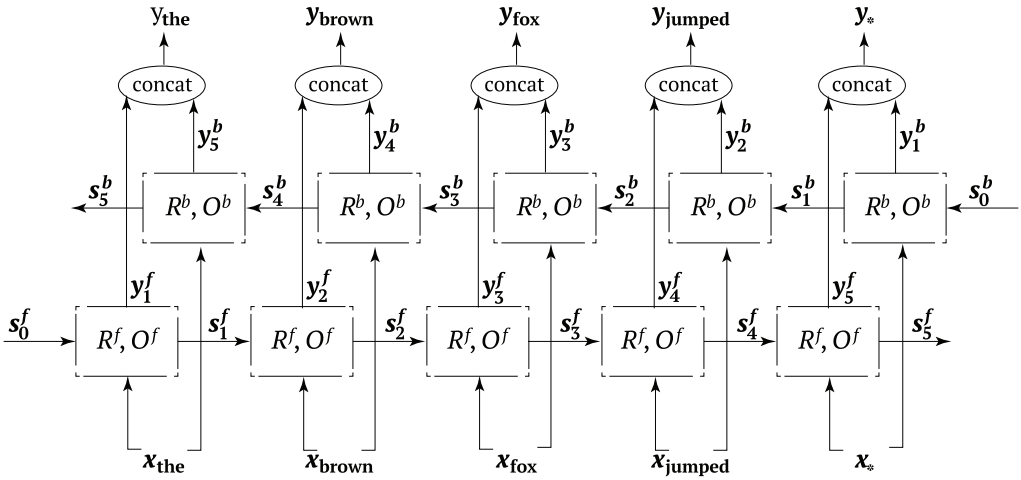


Рис. 14.6 ❖ Вычисление biRNN* для предложения «the brown fox jumped»

biRNN очень эффективна для задач разметки, когда каждому входному вектору соответствует один выходной. Она полезна также как обучаемый компонент общего назначения для выделения признаков, который можно использовать всякий раз, как требуется окно вокруг заданного слова. Конкретные примеры использования приведены в главе 16.

Применение biRNN для разметки последовательностей было представлено сообществу NLP в работе Irsoy and Cardie [2014].

14.5. Многослойные РНС

РНС можно располагать в несколько слоев, образующих сетку [Hihi and Bengio, 1996]. Рассмотрим k РНС, $\text{RNN}_1, \ldots, \text{RNN}_k$, где j-я РНС имеет состояния $\boldsymbol{s}^j_{1:n}$ и выходы $\boldsymbol{y}^j_{1:n}$. На вход первой РНС подается последовательность $\boldsymbol{x}_{1:n}$, а на вход j-й РНС ($j \geq 2$) – выходы РНС, расположенной под ней, $\boldsymbol{y}^{j-1}_{1:n}$. Выходом всей конструкции является выход последней РНС, $\boldsymbol{y}^k_{1:n}$. Такие многослойные архитектуры часто называют *глубокими РНС*. Трехслойная РНС показана на рис. 14.7. Точно так же можно составлять слои из biRNN[1].

[1] Термин *глубокая biRNN* используется в литературе для описания двух разных архитектур. В первом случае состоянием biRNN является конкатенация двух глубоких РНС. Во втором случае выходная последовательность одной biRNN подается на вход другой. Как показали исследования моей группы, второй вариант часто работает лучше.

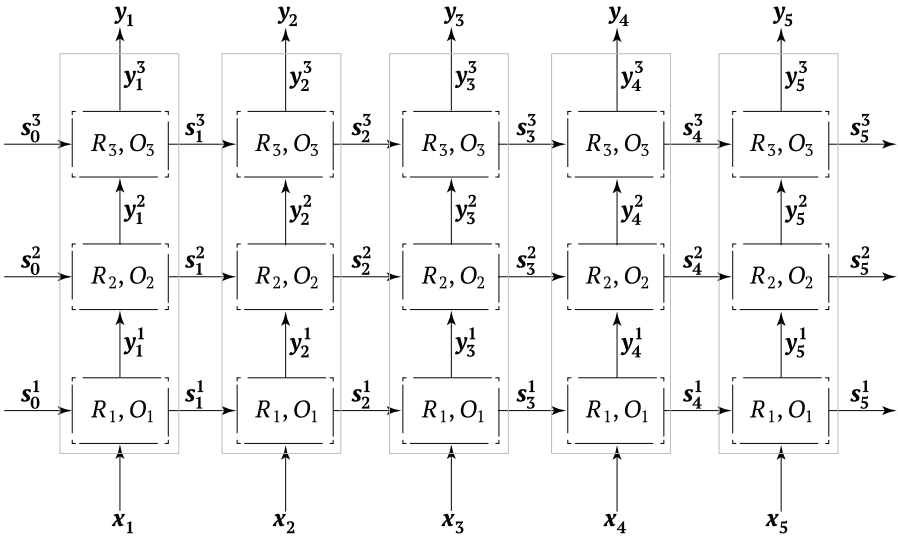


Рис. 14.7 ❖ Трехслойная («глубокая») архитектура РНС

Хотя теоретически не ясно, какие дополнительные преимущества может дать более глубокая архитектура, эмпирически было замечено, что для некоторых задач глубокая РНС работает лучше мелкой. В частности, в работе Sutskever et al. [2014] сообщается, что для достижения хорошего качества машинного перевода в инфраструктуре кодировщик–декодер понадобилась четырехслойная глубокая архитектура. В работе Irsoy and Cardie [2014] также утверждается, что результаты улучшились после перехода от однослойной biRNN к архитектуре с несколькими слоями. Во многих других работах также описываются результаты использования многослойных архитектур РНС, но не проводится явное сравнение с однослойными. В эксперименте моей исследовательской группы применение двух и более слоев зачастую действительно оказывалось лучше, чем один слой.

14.6. РНС для представления стеков

В некоторых алгоритмах лингвистической обработки, в частности для разбора на основе переходов состояний [Nivre, 2008], требуется выполнять выделение признаков из стека. Вместо того чтобы ограничиваться просмотром k верхних элементов стека, РНС можно использовать для того, чтобы закодировать весь стек вектором фиксированного размера.

Основная идея заключается в том, что стек, по существу, представляет собой последовательность, поэтому состояние стека можно представить, если выбирать из него элементы и подавать их по очереди на вход РНС, в результате чего мы в конце получим закодированный стек. Чтобы это вычисление было эффективным (без выполнения $O(n)$ операций кодирования стека при каждом его изме-

нении), состояние РНС запоминается вместе с состоянием стека. Если бы в стек можно было только заталкивать, то все было бы тривиально: при заталкивании каждого нового элемента x в стек мы использовали бы соответствующий вектор $\boldsymbol{x}$ вместе с состоянием РНС $\boldsymbol{s}_i$, чтобы получить новое состояние $\boldsymbol{s}_{i+1}$. Операция выталкивания сложнее, но с ней можно справиться, воспользовавшись структурой данных «неизменяемый стек» (persistent stack) [Goldberg et al., 2013, Okasaki, 1999]. Неизменяемая структура данных сохраняет свои прежние версии при модификации. В неизменяемом стеке стек представляется указателем на первый элемент связного списка. Пустой стек – это пустой список. Операция push добавляет элемент в список и возвращает новый первый элемент. Операция pop возвращает родителя первого элемента, но сохраняет исходный список неизменным. С точки зрения человека, хранящего указатель на прежний первый элемент, стек не изменился. Последующая операция push добавит нового потомка того же узла. Применение этой процедуры в течение всего времени существования стека приводит к созданию дерева, корень которого – пустой стек, а каждый путь от узла к корню представляет некоторое промежуточное состояние стека. На рис. 14.8 приведен пример такого дерева. Тот же процесс можно применить к построению графа вычислений, при этом будет создана РНС с древовидной, а не линейной структурой. Обратное распространение ошибки из данного узла затронет все элементы, присутствовавшие в стеке на момент создания этого узла, по порядку. На рис. 14.9 показан граф вычислений для стековой РНС, соответствующей последнему состоянию на рис. 14.8. Такой подход к моделированию был независимо предложен в работах Dyer et al. [2015] и Watanabe and Sumita [2015] для разбора зависимостей на основе переходов состояний.

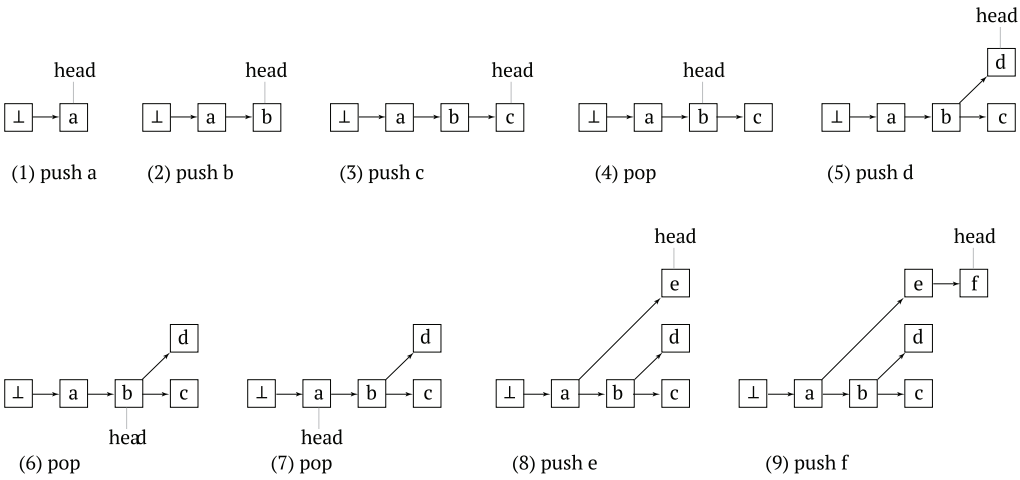


Рис. 14.8 ❖ Построение неизменяемого стека для последовательности операций *push a*; *push b*; *push c*; *pop*; *push d*; *pop*; *pop*; *push e*; *push f*

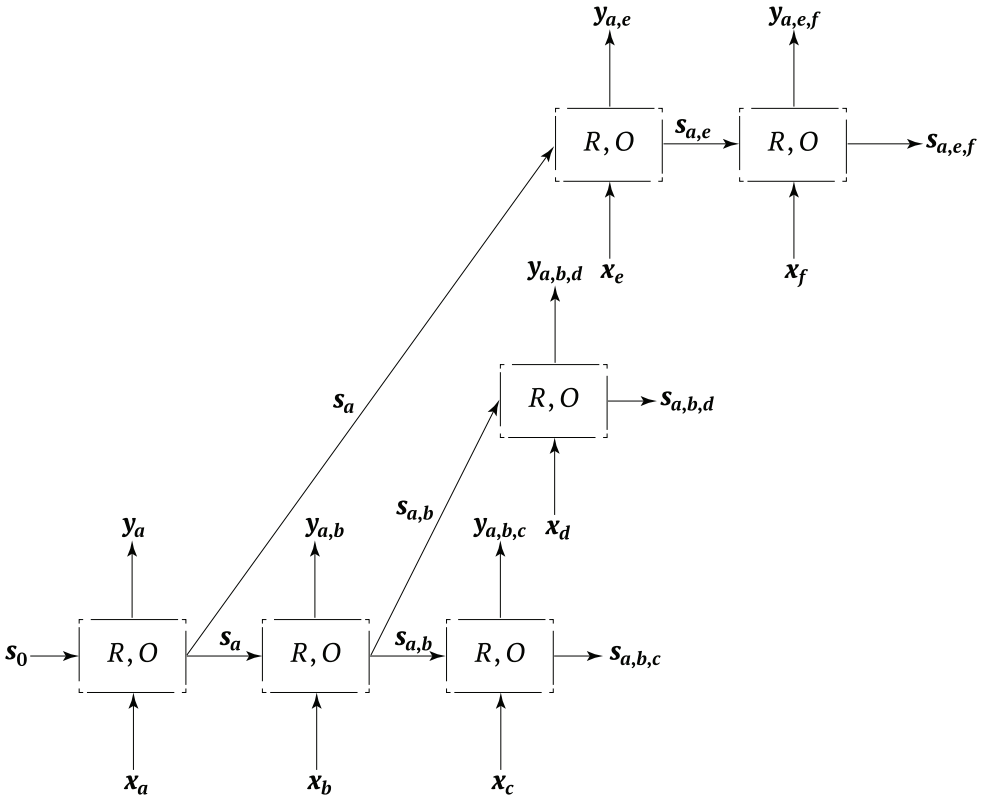


Рис. 14.9 ❖ Стековая РНС, соответствующая последнему состоянию на рис. 14.8

14.7. Замечание о чтении литературы

К сожалению, часто бывает очень трудно понять, как устроена модель, читая ее описание в научной статье. Многие аспекты моделей еще не стандартизованы, и разные исследователи называют одним словом совершенно разные вещи. Приведем лишь несколько примеров. Входами РНС могут быть унитарные векторы (в таком случае матрица погружений является внутренней частью РНС) или представления в виде погружений; входная последовательность может дополняться символами начала и (или) конца последовательности либо не дополняться; обычно предполагается, что выходом РНС является вектор, который нужно будет подать на вход дополнительных слоев, за которыми следует вызов softmax для получения предсказания, но в некоторых работах считается, что softmax – это часть самой РНС; в многослойных РНС «вектором состояния» может быть как выход верхнего слоя, так и конкатенация выходов всех слоев; при использовании каркаса кодировщик–декодер обусловливание выходом кодировщика может интерпретироваться разными способами и т. д. Как будто этого недостаточно, у архитектуры LSTM, описанной в следующей главе, полно мелких вариаций, но все они

называются LSTM. Иногда тонкости такого рода явно упомянуты в тексте статьи, бывает, что для их осознания нужно внимательно читать, а бывает и так, что нюансы даже не упомянуты или скрыты за неоднозначными рисунками или словесными оборотами.

Читателю необходимо знать об этих проблемах при чтении и интерпретации описания моделей. Но и выступая в роли автора, не забывайте о них: либо полностью опишите модель в математических обозначениях, либо сошлитесь на источник, в котором модель полностью описана, если таковой существует. Если вы используете реализацию по умолчанию из какого-то программного пакета, не зная о деталях, явно отметьте этот факт и укажите, какой пакет использовали. В любом случае не полагайтесь только на рисунки или описание модели на естественном языке, поскольку то и другое чревато неоднозначностью.

Глава 15

Конкретные архитектуры рекуррентных нейронных сетей

Описав абстракцию РНС, мы теперь можем обсудить ее конкретные реализации. Напомним, что нас интересует рекурсивная функция $\boldsymbol{s}_i = R(\boldsymbol{x}_i, \boldsymbol{s}_{i-1})$ – такая, что $\boldsymbol{s}_i$ кодирует последовательность $\boldsymbol{x}_{1:n}$. Мы представим несколько конкретных реализаций абстрактной архитектуры РНС и определения конкретных функций R и O, а именно: *простая РНС* (Simple RNN – S-RNN), *долгая краткосрочная память* (Long Short-Term Memory – LSTM) и *вентильный рекуррентный блок* (Gated Recurrent Unit – GRU).

15.1. CBOW как РНС

Простейшим вариантом функции R является сложение:

$$\begin{aligned} \boldsymbol{s}_i &= R_{\text{CBOW}}(\boldsymbol{x}_i, \boldsymbol{s}_{i-1}) = \boldsymbol{s}_{i-1} + \boldsymbol{x}_i; \\ \boldsymbol{y}_i &= O_{\text{CBOW}}(\boldsymbol{s}_i) = \boldsymbol{s}_i; \\ \boldsymbol{s}_i, \boldsymbol{y}_i &\in \mathbb{R}^{d_s}, \quad \boldsymbol{x}_i \in \mathbb{R}^{d_s}. \end{aligned} \tag{15.1}$$

Определение (15.1) дает нам модель непрерывного мешка слов: состояние, соответствующее входам $\boldsymbol{x}_{1:n}$, равно сумме этих входов. При всей простоте эта реализация РНС игнорирует последовательную природу данных. Описанная ниже РНС Элмана добавляет зависимость от порядка следования элементов[1].

15.2. Простая РНС

Простейшая РНС, чувствительная к порядку элементов последовательности, известна под названием сети Элмана, или простой РНС (S-RNN). Модель S-RNN

[1] Взгляд на CBOW-представление как на РНС нечасто встречается в литературе. Однако нам кажется, что это неплохая промежуточная ступень к определению РНС Элмана. Кроме того, полезно включить простой CBOW-кодировщик в тот же каркас, что и РНС, поскольку он также может играть роль кодировщика в сетях условной генерации, описанных в главе 17.

предложена в работе Elman [1990], а ее применение для языкового моделирования изучалось в работе Mikolov [2012]. S-RNN имеет вид:

$$\begin{aligned} \boldsymbol{s}_i &= R_{\text{SRNN}}(\boldsymbol{x}_i, \boldsymbol{s}_{i-1}) = g(\boldsymbol{s}_{i-1}\boldsymbol{W}^{\boldsymbol{s}} + \boldsymbol{x}_i\boldsymbol{W}^{\boldsymbol{x}} + \boldsymbol{b}); \\ \boldsymbol{y}_i &= O_{\text{SRNN}}(\boldsymbol{s}_i) = \boldsymbol{s}_i, \\ \boldsymbol{s}_i, \boldsymbol{y}_i \in \mathbb{R}^{d_s},\ \ \boldsymbol{x}_i \in \mathbb{R}^{d_x},\ \ \boldsymbol{W}^{\boldsymbol{x}} \in \mathbb{R}^{d_x \times d_s},\ \ \boldsymbol{W}^{\boldsymbol{s}} \in \mathbb{R}^{d_s \times d_s},\ \ \boldsymbol{b} \in \mathbb{R}^{d_s}. \end{aligned} \tag{15.2}$$

То есть состояние $\boldsymbol{s}_{i-1}$ и вход $\boldsymbol{x}_i$ подвергаются линейному преобразованию, результаты складываются (вместе с членом смещения) и сумма передается нелинейной функции активации g (обычно tanh или ReLU). Выход в позиции i совпадает со скрытым состоянием в этой позиции[1].

Уравнения (15.2) можно переписать в эквивалентной форме (15.3), и обе формы встречаются в литературе:

$$\begin{aligned} \boldsymbol{s}_i &= R_{\text{SRNN}}(\boldsymbol{x}_i, \boldsymbol{s}_{i-1}) = g([\boldsymbol{s}_{i-1}; \boldsymbol{x}_i]\boldsymbol{W} + \boldsymbol{b}); \\ \boldsymbol{y}_i &= O_{\text{SRNN}}(\boldsymbol{s}_i) = \boldsymbol{s}_i, \\ \boldsymbol{s}_i, \boldsymbol{y}_i \in \mathbb{R}^{d_s},\ \ \boldsymbol{x}_i \in \mathbb{R}^{d_x},\ \ \boldsymbol{W} \in \mathbb{R}^{(d_x \times d_s) \times d_s},\ \ \boldsymbol{b} \in \mathbb{R}^{d_s}. \end{aligned} \tag{15.3}$$

S-RNN лишь немногим сложнее CBOW, а основное различие между ними – нелинейная функция активации g. Однако это различие критическое, поскольку добавление линейного преобразования с последующей нелинейностью делает сеть чувствительной к порядку входных данных. И действительно, простая РНС дает сильные результаты в задаче разметки предложения [Xu et al., 2015] и в языковом моделировании. Подробное обсуждение использования простых РНС для языкового моделирования *см*. в докторской диссертации Mikolov [2012].

15.3. Вентильные архитектуры

S-RNN трудно обучить эффективно из-за проблемы исчезающих градиентов [Pascanu et al., 2012]. Сигналы ошибки (градиенты) от поздних шагов последовательности быстро убывают в процессе обратного распространения и не достигают более ранних входных сигналов, поэтому S-RNN с трудом улавливает дальние зависимости. Архитектуры на базе вентилей, например LSTM [Hochreiter and Schmidhuber, 1997] и GRU [Cho et al., 2014b], призваны устранить этот недостаток.

Рассмотрим РНС как универсальное вычислительное устройство, в котором состояние $\boldsymbol{s}_i$ представляет конечную память. Каждый вызов функции R читает вход $\boldsymbol{x}_{i+1}$ и текущее состояние памяти $\boldsymbol{s}_i$, производит над ними некоторые действия и записывает результат в память, так что получается новое состояние памяти $\boldsymbol{s}_{i+1}$. При таком взгляде становится понятна проблема архитектуры S-RNN – доступ к памяти не контролируется. На каждом шаге вычисления читается и записывается все состояние памяти.

Как обеспечить более контролируемый доступ к памяти? Рассмотрим двоичный вектор $\boldsymbol{g} \in \{0, 1\}^n$. Такой вектор может работать как *вентиль*, управляющий

[1] Некоторые авторы считают выходом в позиции i более сложную функцию от состояния, например линейное преобразование или МСП. В нашем изложении такое последующее преобразование выхода рассматривается не как часть РНС, а как отдельное вычисление, применяемое к выходу РНС.

доступом к n-мерным векторам с помощью произведения Адамара $\boldsymbol{x} \odot \boldsymbol{g}$[1]. Будем рассматривать память $\boldsymbol{s} \in \mathbb{R}^d$ как вход $\boldsymbol{x} \in \mathbb{R}^d$ и вентиль $\boldsymbol{g} \in \{0, 1\}^d$. Вычисление $\boldsymbol{s'} \leftarrow \boldsymbol{g} \odot \boldsymbol{x} + (1 - \boldsymbol{g}) \odot \boldsymbol{s}$ «читает» элементы $\boldsymbol{x}$, соответствующие единичным элементам $\boldsymbol{g}$, и записывает их в новую память $\boldsymbol{s'}$. Затем ячейки, которые не были прочитаны, копируются из $\boldsymbol{s}$ в новую память $\boldsymbol{s'}$ благодаря использованию вентиля $(1 - \boldsymbol{g})$. На рис. 15.1 изображен процесс обновления памяти в позициях входа 2 и 5.

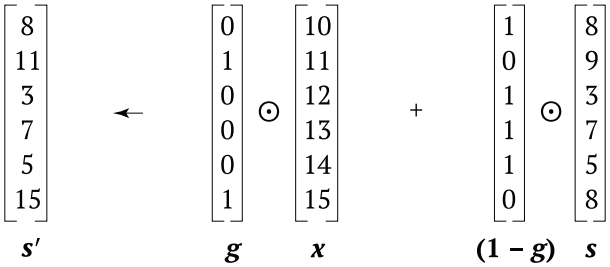


Рис. 15.1 ❖ Использование двоичного вентильного вектора $\boldsymbol{g}$ для управления доступом к памяти $\boldsymbol{s'}$

Описанный вентильный механизм может послужить строительным блоком в нашей РНС, если использовать вентильные векторы для управления доступом к состоянию памяти $\boldsymbol{s}_i$. Однако нам все еще не хватает двух важных (и взаимосвязанных) вещей: вентили должны быть не статическими, а управляемыми текущим состоянием памяти и входными данными, а их поведение должно поддаваться обучению. Это создает препятствие, поскольку при нашем подходе к обучению требуется дифференцируемость (из-за алгоритма обратного распространения), а вентили, принимающие двоичные значения 0 и 1, не дифференцируемы[2].

Возникшую проблему можно решить, аппроксимировав жесткий вентильный механизм мягким, но дифференцируемым. Чтобы получить такие *дифференцируемые вентили*, мы откажемся от требования $\boldsymbol{g} \in \{0, 1\}^n$ и разрешим использовать произвольные вещественные числа $\boldsymbol{g'} \in \mathrm{R}^n$, которые будут подаваться на вход сигмоидной функции $\sigma(\boldsymbol{g'})$. Эта функция приводит любое значение к диапазону (0, 1), причем большая часть значений концентрируется близко к границам диапазона. При использовании вентиля $\sigma(\boldsymbol{g'}) \odot \boldsymbol{x}$ индексы $\boldsymbol{x}$, соответствующие близким к единице значениям $\sigma(\boldsymbol{g'})$, пропускаются, а соответствующие значениям, близким к нулю, – блокируются. Тогда значения вентиля можно обусловить входом и текущим состоянием памяти и обучать градиентными методами для достижения желаемого поведения.

Этот управляемый вентильный механизм и является основой архитектур LSTM и GRU, которые мы определим ниже: на каждом временном шаге дифференцируемый вентильный механизм решает, какую часть входов записать в память

---

[1] Произведение Адамара – это научное название поэлементного умножения двух векторов: $\boldsymbol{x} = \boldsymbol{u} \odot \boldsymbol{v}$ дает вектор с элементами $\boldsymbol{x}_{[i]} = \boldsymbol{u}_{[i]} \cdot \boldsymbol{v}_{[i]}$.

[2] В принципе, возможно обучать и модели с недифференцируемыми компонентами, в том числе двоичные вентили, если воспользоваться методами обучения с подкреплением. Однако на момент написания этой книги такие методы были еще не слишком надежны. Обучение с подкреплением выходит за рамки этой книги.

и какую часть памяти перезаписать (забыть). Это довольно абстрактное описание мы конкретизируем в следующих разделах.

15.3.1. LSTM

Архитектура долгой краткосрочной памяти (Long Short-Term Memory – LSTM) [Hochreiter and Schmidhuber, 1997] разрабатывалась для решения проблемы исчезающих градиентов и стала первым примером вентильного механизма. В этой архитектуре вектор состояния явным образом разбивается на две половины, одна из которых рассматривается как «ячейки памяти», а другая является рабочей памятью. Ячейки предназначены для сохранения памяти и градиентов ошибки на протяжении времени и управляются *дифференцируемыми вентильными компонентами* – гладкими математическими функциями, которые моделируют логические вентили. В каждом входном состоянии вентиль используется для принятия решения о том, какую часть новых входных данных записать в ячейку памяти и какую часть текущего содержания ячеек памяти забыть. Математически архитектура LSTM определяется следующим образом[1]:

$$
\begin{aligned}
\boldsymbol{s}_j = R_{\text{LSTM}}(\boldsymbol{s}_{j-1}, \boldsymbol{x}_j) &= [\boldsymbol{c}_j; \boldsymbol{h}_j];\\
\boldsymbol{c}_j &= \boldsymbol{f} \odot \boldsymbol{c}_{j-1} + \boldsymbol{i} \odot \boldsymbol{z};\\
\boldsymbol{h}_j &= \boldsymbol{o} \odot \tanh(\boldsymbol{c}_j);\\
\boldsymbol{i} &= \sigma(\boldsymbol{x}_j \boldsymbol{W}^{\boldsymbol{xi}} + \boldsymbol{h}_{j-1} \boldsymbol{W}^{\boldsymbol{hi}});\\
\boldsymbol{f} &= \sigma(\boldsymbol{x}_j \boldsymbol{W}^{\boldsymbol{xf}} + \boldsymbol{h}_{j-1} \boldsymbol{W}^{\boldsymbol{hf}});\\
\boldsymbol{o} &= \sigma(\boldsymbol{x}_j \boldsymbol{W}^{\boldsymbol{xo}} + \boldsymbol{h}_{j-1} \boldsymbol{W}^{\boldsymbol{ho}});\\
\boldsymbol{z} &= \tanh(\boldsymbol{x}_j \boldsymbol{W}^{\boldsymbol{xz}} + \boldsymbol{h}_{j-1} \boldsymbol{W}^{\boldsymbol{hz}});\\
\boldsymbol{y}_j = O_{\text{LSTM}}(\boldsymbol{s}_j) &= \boldsymbol{h}_j,\\
\boldsymbol{s}_j \in \mathbb{R}^{2 \cdot d_h},\ \boldsymbol{x}_i \in \mathbb{R}^{d_x},\ \boldsymbol{c}_j, \boldsymbol{h}_j, \boldsymbol{i}, \boldsymbol{f}, \boldsymbol{o}, \boldsymbol{z} \in \mathbb{R}^{d_h},\ \boldsymbol{W}^{\boldsymbol{xo}} \in \mathbb{R}^{d_x \times d_h},\ \boldsymbol{W}^{\boldsymbol{ho}} \in \mathbb{R}^{d_h \times d_h}.&
\end{aligned}
\tag{15.4}
$$

Состояние в момент j образовано двумя векторами, $\boldsymbol{c}_j$ и $\boldsymbol{h}_j$, где $\boldsymbol{c}_j$ – компонент памяти, а $\boldsymbol{h}_j$ – компонент скрытого состояния. Имеются три вентиля, $\boldsymbol{i}$, $\boldsymbol{f}$ и $\boldsymbol{o}$, управляющие соответственно входом, забыванием и выходом. Значения вентилей вычисляются как результат применения сигмоидной функции активации к линейной комбинации текущего входа $\boldsymbol{x}_j$ и предыдущего состояния $\boldsymbol{h}_{j-1}$. Предлагаемое обновление $\boldsymbol{z}$ вычисляется как результат применения функции активации tanh к линейной комбинации $\boldsymbol{x}_j$ и $\boldsymbol{h}_{j-1}$. Затем обновляется память $\boldsymbol{c}_j$: вентиль забывания управляет тем, какую часть предыдущей памяти сохранить ($\boldsymbol{f} \odot \boldsymbol{c}_{j-1}$), а входной вентиль – какую часть предлагаемого обновления сохранить ($\boldsymbol{i} \odot \boldsymbol{z}$). Наконец, значение $\boldsymbol{h}_j$ (совпадающее с выходом $\boldsymbol{y}_j$) определяется на основе содержимого памяти $\boldsymbol{c}_j$, пропущенного через нелинейность tanh, и управляется выходным вентилем. Вентильный механизм обеспечивает достаточно большие значения градиентов, относящихся к части памяти $\boldsymbol{c}_j$, на протяжении очень длительного времени.

---

[1] Существует много вариантов представленной здесь архитектуры LSTM. Например, вентилей забывания не было в оригинальной статье Hochreiter and Schmidhuber [1997], но затем было продемонстрировано, что это важная часть архитектуры. Из других вариантов упомянем связи с глазками и связывание вентилей. Обзор и результаты эмпирического сравнения различных архитектур LSTM *см.* в работе Greff et al. [2015].

Дополнительные сведения об архитектуре LSTM *см*. в докторской диссертации Alex Graves [2008], а также в описании Криса Ола[1]. Анализ поведения LSTM при использовании в качестве языковой модели на уровне литер *см*. в работе Karpathy et al. [2015].

ПРОБЛЕМА ИСЧЕЗАЮЩИХ ГРАДИЕНТОВ В РЕКУРРЕНТНЫХ НЕЙРОННЫХ СЕТЯХ И ЕЕ РЕШЕНИЕ Интуитивно понятно, что рекуррентную нейронную сеть можно рассматривать как очень глубокую сеть прямого распространения, параметры которой разделяются между разными слоями. Поэтому для простой РНС (15.3) вычисление градиентов включает повторное умножение на матрицу $\boldsymbol{W}$, из-за чего велика вероятность обнуления или взрывного роста значений. Вентильный механизм смягчает эту проблему за счет избавления от повторного умножения на одну и ту же матрицу.
Дальнейшее обсуждение проблемы исчезающих и взрывных градиентов в РНС *см*. в разделе 10.7 работы Bengio et al. [2016]. Более подробное объяснение мотивов введения вентильного механизма в LSTM (и GRU) и его связи с решением проблемы исчезающих градиентов *см*. в разделах 4.2 и 4.3 детальных заметок к курсу Cho [2015].

В настоящее время LSTM – самый успешный вид архитектуры РНС, благодаря ей получены многие из лучших на сегодняшний момент результатов в моделировании последовательностей. Основным конкурентом LSTM является архитектура GRU, которую мы рассмотрим ниже.

ПРАКТИЧЕСКИЕ СООБРАЖЕНИЯ В работе Jozefowicz et al. [2015] настоятельно рекомендуется при обучении LSTM-сетей инициализировать член смещения в вентиле забывания значением, близким к 1.

15.3.2. GRU

Архитектура LSTM очень эффективна, но при этом весьма сложна. Сложность затрудняет анализ системы и дорого обходится с вычислительной точки зрения. Вентильный рекуррентный блок (GRU), относительно недавно предложенный в работе Cho et al. [2014b], является альтернативой LSTM. Впоследствии в работе Chung et al. [2014] было показано, что эта архитектура дает результаты, сравнимые с LSTM, на нескольких (не текстовых) наборах данных.

Как и LSTM, архитектура GRU основана на вентильном механизме, но вентилей существенно меньше и отдельного компонента памяти не существует:

$$\begin{aligned} \boldsymbol{s}_j = R_{\text{GRU}}(\boldsymbol{s}_{j-1}, \boldsymbol{x}_j) &= (\mathbf{1} - \boldsymbol{z}) \odot \boldsymbol{s}_{j-1} + \boldsymbol{z} \odot \tilde{\boldsymbol{s}}_j]; \\ \boldsymbol{z} &= \sigma(\boldsymbol{x}_j \boldsymbol{W}^{\boldsymbol{xz}} + \boldsymbol{s}_{j-1} \boldsymbol{W}^{\boldsymbol{sz}}); \\ \boldsymbol{r} &= \sigma(\boldsymbol{x}_j \boldsymbol{W}^{\boldsymbol{xr}} + \boldsymbol{s}_{j-1} \boldsymbol{W}^{\boldsymbol{sr}}); \\ \tilde{\boldsymbol{s}}_j &= \tanh(\boldsymbol{x}_j \boldsymbol{W}^{\boldsymbol{xs}} + (\boldsymbol{r} \odot \boldsymbol{s}_{j-1}) \boldsymbol{W}^{\boldsymbol{sg}}); \\ \boldsymbol{y}_j = O_{\text{GRU}}(\boldsymbol{s}_j) &= \boldsymbol{s}_j, \end{aligned} \tag{15.5}$$

$$\boldsymbol{s}_j, \tilde{\boldsymbol{s}}_j \in \mathbb{R}^{d_s}, \quad \boldsymbol{x}_i \in \mathbb{R}^{d_x}, \quad \boldsymbol{z}, \boldsymbol{r} \in \mathbb{R}^{d_s}, \quad \boldsymbol{W}^{\boldsymbol{xo}} \in \mathbb{R}^{d_x \times d_s}, \quad \boldsymbol{W}^{\boldsymbol{so}} \in \mathbb{R}^{d_s \times d_s}.$$

[1] http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Один вентиль ($\boldsymbol{r}$) служит для управления доступом к предыдущему состоянию $\boldsymbol{s}_{j-1}$ и вычисляет предлагаемое обновление $\tilde{\boldsymbol{s}}_j$. Затем обновленное состояние $\boldsymbol{s}_j$ (совпадающее с выходом $\boldsymbol{y}_j$) вычисляется путем интерполяции предыдущего состояния $\boldsymbol{s}_{j-1}$ и предложения $\tilde{\boldsymbol{s}}_j$, причем коэффициенты интерполяции управляются вентилем $\boldsymbol{z}$[1].

Показано, что архитектура GRU эффективна в задачах языкового моделирования и машинного перевода. Однако все еще не сделан окончательный выбор между GRU, LSTM и возможными альтернативными архитектурами РНС, это тема активных исследований. Результаты эмпирического изучения архитектур GRU и LSTM *см*. в работе Jozefowicz et al. [2015].

15.4. Другие варианты

УСОВЕРШЕНСТВОВАНИЯ НЕВЕНТИЛЬНЫХ АРХИТЕКТУР Вентильные архитектуры LSTM и GRU помогают сгладить проблему исчезающих градиентов, характерную для простой РНС, и улавливают долгосрочные зависимости. Некоторые исследователи применяли более простые архитектуры для получения аналогичных преимуществ.

В работе Mikolov et al. [2014] подмечено, что умножение на матрицу $\boldsymbol{s}_{i-1}\boldsymbol{W}^{\boldsymbol{s}}$ в сочетании с нелинейностью $\boldsymbol{g}$ в правиле обновления R для простой РНС приводит к тому, что вектор состояния $\boldsymbol{s}_i$ сильно изменяется на каждом временном шаге, что мешает ему запоминать информацию на протяжении длительных периодов времени. Авторы предлагают разделить вектор состояния $\boldsymbol{s}_i$ на медленно изменяющуюся компоненту $\boldsymbol{c}_i$ («контекстные блоки») и быстро изменяющуюся компоненту $\boldsymbol{h}_i$[2]. Медленно изменяющаяся компонента $\boldsymbol{c}_i$ обновляется по правилу линейной интерполяции входа и предыдущего значения компоненты: $\boldsymbol{c}_i = (1-\alpha)\boldsymbol{x}_i\boldsymbol{W}^{x1} + \alpha\boldsymbol{c}_{i-1}$, где $\alpha \in (0, 1)$. Это обновление позволяет $\boldsymbol{c}_i$ аккумулировать предыдущие входы. Быстро изменяющаяся компонента $\boldsymbol{h}_i$ обновляется по правилу, похожему на применяющееся в простой РНС, но модифицированному с учетом $\boldsymbol{c}_i$[3]: $\boldsymbol{h}_i = \sigma(\boldsymbol{x}_i\boldsymbol{W}^{x2} + \boldsymbol{h}_{i-1}\boldsymbol{W}^{h} + \boldsymbol{c}_i\boldsymbol{W}^{c})$. Наконец, выходом $\boldsymbol{y}_i$ является конкатенация медленно и быстро изменяющейся компоненты состояния: $\boldsymbol{y}_i = [\boldsymbol{c}_i; \boldsymbol{h}_i]$. Миколов с коллегами продемонстрировали, что в задачах языкового моделирования перплексивность этой архитектуры вполне может конкурировать с той, что дает гораздо более сложная LSTM.

Подход Миколова с коллегами можно интерпретировать как ограничение, наложенное на блок матрицы $\boldsymbol{W}^{\boldsymbol{s}}$ в простой РНС, соответствующей $\boldsymbol{c}_i$: он должен быть кратным единичной матрице (детали *см*. в работе Mikolov et al. [2014]). В работе Le et al. [2015] предложен еще более простой подход: использовать ReLU в роли функции активации простой РНС и инициализировать смещения $\boldsymbol{b}$ нулями, а в качестве $\boldsymbol{W}^{\boldsymbol{s}}$ взять единичную матрицу. Это заставляет необученную РНС

---

[1] В литературе по GRU состояние $\boldsymbol{s}$ часто обозначают $\boldsymbol{h}$.

[2] Мы отклонились от обозначений, принятых в работе Mikolov et al. [2014], и повторно используем те же символы, что в описании LSTM.

[3] Это правило обновления отличается от применяемого в S-RNN еще и тем, что в качестве нелинейности берется только сигмоидная функция, а член смещения не используется вовсе. Однако не эти изменения считаются основными.

скопировать предыдущее состояние в текущее, прибавить влияние текущего входа $\boldsymbol{x}_t$ и заменить отрицательные значения нулями. После такой установки предпочтения копированию состояния процедура обучения позволяет $\boldsymbol{W}^s$ изменяться как угодно. Ле с коллегами продемонстрировал, что в результате этой нехитрой модификации простая РНС оказывается сравнима с LSTM с таким же числом параметров на некоторых задачах, в том числе языкового моделирования.

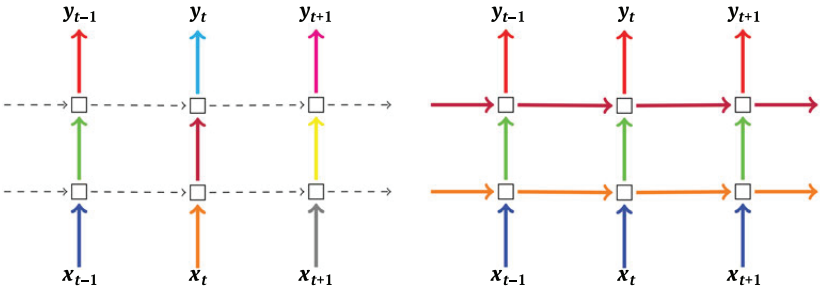
НЕ ТОЛЬКО ДИФФЕРЕНЦИРУЕМЫЕ ВЕНТИЛИ Вентильный механизм – пример адаптации понятий теории вычислений (доступ к памяти, логические вентили) к дифференцируемым – и, стало быть, допускающим градиентное обучение – системам. Исследователи проявляют большой интерес к созданию архитектур нейронных сетей для имитационного моделирования и реализации других вычислительных механизмов, которые допускают улучшенное и более точное управление. Два примера такого рода – работа по *дифференцируемому стеку* [Grefenstette et al., 2015], в которой структура стека с операциями push и pop управляется дифференцируемой сетью, и *нейронная машина Тьюринга* [Graves et al., 2014], которая допускает доступ для чтения и записи к ассоциативной памяти в дифференцируемой системе. Хотя эти попытки пока не привели к созданию надежных универсальных архитектур, которые можно было бы использовать в реальных приложениях лингвистической обработки, стоит поглядывать в этом направлении.

15.5. Прореживание в РНС

Применение прореживания к РНС не вполне тривиально, поскольку отбрасывание различных измерений на разных временных шагах подавляет способность РНС передавать информативные сигналы во времени. Поэтому в работах Pham et al. [2013], Zaremba et al. [2014] было предложено применять прореживание только к нерекурсивной связи, т. е. между слоями в глубоких РНС, а не между позициями последовательности.

Позже в работе Gal [2015], опирающейся на результаты вариационного анализа архитектуры РНС, было предложено применять прореживание ко всем компонентам РНС (рекуррентным и нерекуррентным), но сохранять одну и ту же маску прореживания на всех временных шагах. То есть случайный выбор масок прореживания производится один раз на всю последовательность, а не на каждом временном шаге. На рис. 15.2 эта форма прореживания («вариационная РНС») сравнивается с архитектурой из работ Pham et al. [2013], Zaremba et al. [2014].

Вариационная РНС Гэла в настоящее время считается наилучшим методом прореживания в РНС.



(a) RNN с наивным прореживанием (b) Вариационная RNN

Рис. 15.2 ❖ Сравнение предложенного Гэлом метода прореживания РНС (b) с более ранним предложением из работ Pham et al. [2013], Zaremba et al. [2014] (a). Рисунок взят из работы Gal [2015] с разрешения авторов. Каждый квадратик представляет один блок РНС, горизонтальные стрелки представляют временную зависимость (рекуррентные связи), а вертикальные стрелки – входы и выходы блоков РНС. Цветные связи – это прореженные входы, причем разные цвета соответствуют разным маскам прореживания. Штриховые линии соответствуют стандартным связям без прореживания. В предшествующих методах (наивное прореживание слева) использовались разные маски на разных временных шагах, без прореживания в рекуррентных слоях. В методе Гэла (вариационная РНС справа) на каждом временном шаге, включая и рекуррентные слои, используется одна и та же маска

Глава 16

Моделирование с помощью рекуррентных сетей

В главе 14 мы перечислили типичные способы использования РНС, в главе 15 описали детали конкретных архитектур, а теперь готовы проиллюстрировать использование РНС в приложениях NLP на конкретных примерах. Хотя мы употребляем общий термин РНС, обычно имеются в виду вентильные архитектуры типа LSTM или GRU. Простая РНС неизменно дает меньшую верность.

16.1. Приемщики

Проще всего использовать РНС в качестве приемщика: прочитать входную последовательность и породить в конце бинарный или многоклассовый ответ. РНС являются очень сильными обучаемыми и могут улавливать чрезвычайно тонкие паттерны в последовательных данных.

Такая изощренность излишня для многих задач классификации естественного языка: порядок слов и структура предложения зачастую оказываются не слишком важны, так что классификатор на основе мешка слов или мешка *n*-грамм дает результаты не хуже, чем РНС-приемщики, а то и лучше.

В этом разделе приведено два примера использования приемщиков в лингвистических проблемах. Первый пример канонический: классификация по эмоциональной окраске. Решение на основе РНС работает хорошо, но менее мощные подходы тоже вполне конкурентоспособны. Второй пример несколько искусственный: в нем не решается никакой «полезной» задачи, но демонстрируется мощь РНС и характер распознаваемых ими паттернов.

16.1.1. Классификация по эмоциональной окраске

Классификация по эмоциональной окраске на уровне предложения

В задаче классификации по эмоциональной окраске на уровне предложения дано предложение, нередко в виде части отзыва, и требуется сопоставить ему одно из двух значений: положительное или отрицательное[1]. Это несколько упрощенный подход к задаче определения эмоциональной окраски, но тем не менее он часто используется на практике. Та же задача привела нас к обсуждению сверточных

[1] В более сложном варианте классификация трехзначная: положительное, отрицательное и нейтральное.

нейронных сетей в главе 13. Вот два примера положительного и отрицательного предложений, естественно возникающих в отзывах на кинофильмы[1]:

Положительное: *It's not life-affirming – it's vulgar and mean, but I liked it* (Он не жизнеутверждающий – вульгарный и гнусный, но мне он нравится).

Отрицательное: *It's a disappointing that it only manages to be decent instead of dead brilliant* (Меня разочаровало, что ему всего лишь удалось стать достойным, а не самым лучшим).

Обратите внимание, что положительный пример содержит отрицательные фразы (*not life affirming*, *vulgar and mean*), тогда как отрицательный содержит положительные фразы (*dead brilliant*). Для правильного предсказания эмоциональной окраски нужно понимать не только отдельные фразы, но и окружающий их контекст, лингвистические конструкции, в частности отрицание, и общую структуру предложения. Классификация по эмоциональной окраске – трудная и амбициозная задача, правильное решение которой включает учет таких вещей, как сарказм и метафоричность. Само определение эмоциональной окраски далеко не очевидно. Хороший и полный обзор проблем классификации по эмоциональной окраске и определения этого понятия *см*. в работе Pang and Lee [2008]. Но мы не будем обращать внимания на тонкости определения, а будем рассматривать проблему как задачу бинарной классификации, управляемую данными.

Эту задачу легко смоделировать с помощью РНС-приемщика: после лексемизации РНС читает слова предложения по одному. Затем последнее состояние РНС подается на вход МСП, за которым следует слой softmax с двумя выходами. В качестве функции потери при обучении используется перекрестная энтропия, основанная на золотых метках эмоциональной окраски. Для более точной классификации по шкале 1–5 или 1–10 («оценка по числу звездочек») достаточно изменить МСП так, чтобы он выдавал 5 выходов вместо 2. Таким образом получаем следующую архитектуру:

$$\begin{aligned} p(\text{label} = k|w_{1:n}) &= \hat{\boldsymbol{y}}_{[k]}; \\ \hat{\boldsymbol{y}} &= \text{softmax}(\text{MLP}(\text{RNN}(\boldsymbol{x}_{1:n}))); \\ \boldsymbol{x}_{1:n} &= \boldsymbol{E}_{[w_1]}, \ldots, \boldsymbol{E}_{[w_n]}. \end{aligned} \tag{16.1}$$

Матрица погружений слов $\boldsymbol{E}$ инициализируется с помощью погружений, предварительно обученных на большом внешнем корпусе с применением алгоритма типа Word2Vec или GloVe с относительно широким окном.

Часто бывает полезно обобщить модель (16.1), рассмотрев две РНС, одна из которых читает предложение в его естественном порядке, а вторая – в обратном. Конечные состояния обеих РНС конкатенируются и подаются на вход МСП для классификации:

$$\begin{aligned} p(\text{label} = k|w_{1:n}) &= \hat{\boldsymbol{y}}_{[k]}; \\ \hat{\boldsymbol{y}} &= \text{softmax}(\text{MLP}([\text{RNN}^{\text{f}}(\boldsymbol{x}_{1:n}); \text{RNN}^{\text{b}}(\boldsymbol{x}_{n:1})])); \\ \boldsymbol{x}_{1:n} &= \boldsymbol{E}_{[w_1]}, \ldots, \boldsymbol{E}_{[w_n]}. \end{aligned} \tag{16.2}$$

[1] Оба примера взяты из набора данных *Stanford Sentiment Treebank* [Socher et al., 2013b].

Такие двунаправленные модели дают отличные результаты для рассматриваемой задачи [Li et al., 2015].

Для более длинных предложений в работе Li et al. [2015] признано полезным использовать иерархическую архитектуру, в которой предложение разбивается на меньшие отрезки, исходя из знаков препинания. Затем каждый отрезок подается на вход двунаправленной РНС (16.2). Последовательность получившихся векторов (по одному для каждого отрезка) подается на вход РНС-приемщику типа (16.1). Формально пусть дано предложение $w_{1:n}$, разбитое на m отрезков, $w^1_{1:\ell_1}$, ..., $w^m_{1:\ell_m}$. Тогда архитектура описывается уравнениями:

$$\begin{aligned} p(\text{label} = k|w_{1:n}) &= \hat{\boldsymbol{y}}_{[k]}; \\ \hat{\boldsymbol{y}} &= \text{softmax}(\text{MLP}(\text{RNN}(\boldsymbol{z}_{1:m}))); \\ \boldsymbol{z}_i &= [\text{RNN}^{\text{f}}(\boldsymbol{x}^i_{1:\ell_1}); \text{RNN}^{\text{b}}(\boldsymbol{x}^i_{\ell_1:1})]; \\ \boldsymbol{x}^i_{1:\ell_1} &= \boldsymbol{E}_{[w^i_1]}, \ldots, \boldsymbol{E}_{[w^i_{\ell_i}]}. \end{aligned} \tag{16.3}$$

Каждый из m отрезков может быть эмоционально окрашен по-своему. Приемщик верхнего уровня читает сводку $\boldsymbol{z}_{1:m}$, порожденную кодировщиками нижнего уровня, и принимает решение об окраске всего предложения. Классификация по эмоциональной окраске используется также в роли испытательного стенда для иерархических древовидных рекурсивных нейронных сетей, которые описываются в главе 18.

Классификация по эмоциональной окраске на уровне документа

Классификация документа по эмоциональной окраске похожа на классификацию предложения, но входной текст гораздо длиннее – он состоит из нескольких предложений, – а сигнал от учителя (метка эмоциональной окраски) дается только в конце, а не для каждого предложения в отдельности. Эта задача труднее, чем классификация на уровне предложения, поскольку отдельные предложения могут быть окрашены иначе, чем документ в целом.

В работе Tang et al. [2015] установлено, что для этой задачи имеет смысл использовать иерархическую архитектуру, аналогичную описанной в работе Li et al. [2015] [уравнения (16.3)]: каждое предложение кодируется с помощью вентильной РНС, порождающей вектор $\boldsymbol{z}_i$, после чего векторы $\boldsymbol{z}_{1:n}$ подаются на вход второй вентильной РНС, порождающей вектор $\boldsymbol{h} = \text{RNN}(\boldsymbol{z}_{1:n})$, который и используется для предсказания: $\hat{\boldsymbol{y}} = \text{softmax}(\text{MLP}(\boldsymbol{h}))$.

Авторы также экспериментировали с вариантом, в котором все промежуточные векторы РНС уровня документа сохраняются, и их среднее подается на вход МСП ($\boldsymbol{h}_{1:n} = \text{RNN}^*(\boldsymbol{z}_{1:n})$, $\hat{\boldsymbol{y}} = \text{softmax}(\text{MLP}(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{h}_i))$. В некоторых случаях такая архитектура давала чуть лучшие результаты.

16.1.2. Определение грамматической правильности согласования глагола с субъектом

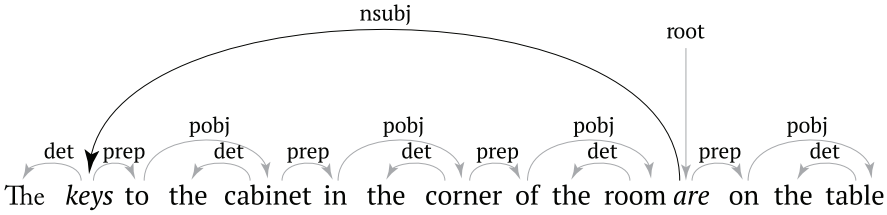
В грамматически правильно построенных английских предложениях заглавное слово субъекта глагола в настоящем времени должно быть согласовано с глаголом по числу (знаком * обозначены грамматически неправильные предложения):

(1) a. The *key is* on the table. (Ключ находится на столе.)
b. *The *key* are on the table. (Ключ находятся на столе.)
c. *The *keys* is on the table. (Ключи находится на столе.)
d. The *keys* are on the table. (Ключи находятся на столе.)

Вывести эту связь только из одной лишь последовательности – нетривиальная задача, поскольку два элемента могут быть разделены сколь угодно длинным сентенциональным материалом, который может содержать существительные другого числа:

(2) a. The keys to the cabinet in the corner of the room are on the table. (Ключи от шкафа в углу комнаты находятся на столе.)
b. *The keys to the cabinet in the corner of the room is on the table. (Ключи от шкафа в углу комнаты находится на столе.)

Учитывая сложность выявления субъекта в линейной последовательности предложения, такие зависимости, как согласование глагола с субъектом, служат аргументом в пользу структурных синтаксических представлений человеческого языка [Everaert et al., 2015]. Действительно, если имеется правильное дерево синтаксического разбора предложения, то выделить связь между глаголом и его субъектом становится тривиально:



В совместной работе с Линценом и Дюпу [Linzen et al., 2016] мы задались целью выяснить, могут ли РНС, являющиеся последовательными обучаемыми, уловить эту в значительной степени синтаксическую регулярность, обучаясь только на последовательностях слов. Для проверки мы подготовили несколько задач предсказания, основываясь на естественных текстах, взятых из Википедии. Одной из задач было определение грамматической правильности: РНС должна была прочитать предложение и в конце решить, является оно грамматически правильным или нет. В нашей конфигурации грамматически правильные предложения содержали глагол в настоящем времени, а грамматически неправильные строились из грамматически правильных путем случайного выбора какого-нибудь глагола в настоящем времени и заменой формы его числа на противоположную[1]. Отметим, что решить эту задачу с помощью мешка слов или мешка *n*-грамм было бы чрезвычайно трудно, поскольку зависимость между глаголом и субъектом вытекает из структуры предложения, которая теряется при переходе к представлению в виде мешка слов, да к тому же может включать произвольное число промежуточных слов *n*.

[1] Несколько технических деталей. Мы определяли глаголы автоматически с помощью частеречного разметчика. Мы использовали словарь из 10 000 самых часто встречающихся в корпусе слов, а слова, отсутствующие в словаре, заменяли автоматически присвоенной им меткой части речи.

Модель обучалась как простой приемщик:

$$\hat{\boldsymbol{y}} = \text{softmax}(\text{MLP}(\text{RNN}(\boldsymbol{E}_{[w_1]}, \ldots, \boldsymbol{E}_{[w_n]})))$$

с перекрестной энтропией в качестве потери. В нашем распоряжении были десятки тысяч обучающих предложений и сотни тысяч тестовых предложений (многие случаи согласования просты, а мы хотели, чтобы тестовый набор содержал достаточное число трудных случаев).

Это трудная задача, в которой учитель принимает участие лишь опосредованно: сигнал от учителя не содержит никакого намека на то, что является причиной грамматической правильности или неправильности. РНС должна сама обучиться понятию числа (тому, что слова в единственном и множественном числах относятся к разным группам), понятию согласования (что форма глагола должна соответствовать форме субъекта) и понятию субъектности (определять, какие из предшествующих глаголу существительных определяют форму глагола). Для идентификации правильного субъекта необходимо научиться выявлять синтаксические маркеры вложенных структур, чтобы пропускать отвлекающие существительные во вложенных предложениях. РНС прекрасно справилась с задачей обучения, ей удалось правильно классифицировать подавляющее большинство (верность $>$ 99 %) тестовых образцов. На действительно трудных образцах, где глагол и его субъект были разделены 4 существительными противоположного числа, РНС все равно достигла верности свыше 80 %. Заметим, что если бы мы обучали сеть эвристике, заключающейся в предсказании числа последнего существительного, то на таких образцах верность составила бы 0 %, а если бы в качестве эвристики было взято число случайно выбранного предшествующего существительного, то верность составила бы 20 %.

Подведем итоги: этот эксперимент демонстрирует способность вентильных РНС к обучению, а также виды тонких паттернов и закономерностей, который они способны улавливать.

16.2. РНС как экстракторы признаков

Основной сценарий использования РНС – гибкие обучаемые экстракторы признаков, способные заменить части более традиционных конвейеров выделения признаков при работе с последовательностями. В частности, РНС – отличная замена оконным экстракторам.

16.2.1. Частеречная разметка

Рассмотрим проблему частеречной разметки в контексте РНС.

ОСТОВ: ГЛУБОКАЯ BIRNN Частеречная разметка – частный случай задачи о разметке последовательности, в которой требуется назначить метку каждому из n входных слов. Это делает biRNN идеальным кандидатом на роль базовой структуры.

Получив предложение, содержащее слова $s = w_{1:n}$, мы преобразуем их во входные векторы $\boldsymbol{x}_{1:n}$, применив функцию выделения признаков $\boldsymbol{x}_i = \phi(s, i)$. Входные векторы подаются на вход глубокой biRNN, которая порождает выходные векторы $\boldsymbol{y}_{1:n} = \text{biRNN}^*(\boldsymbol{x}_{1:n})$. Затем каждый вектор $\boldsymbol{y}_i$ подается на вход МСП, который пред-

сказывает одну из k возможных выходных меток слова. Каждый вектор $\mathbf{y}_i$ сфокусирован на i-й позиции последовательности, но содержит также информацию обо всей последовательности, окружающей эту позицию («бесконечное окно»). В процессе обучения biRNN обучается фокусироваться на аспектах последовательности, информативных для предсказания метки w_i, и кодировать их в векторе $\mathbf{y}_i$.

ОТ СЛОВ К ВХОДНЫМ ДАННЫМ С ПОМОЩЬЮ РНС УРОВНЯ ЛИТЕР Как мы будем отображать слово w_i во входной вектор $\boldsymbol{x}_i$? Одна из возможностей – воспользоваться матрицей погружений, которую можно инициализировать случайными числами или предварительно обучить, например с помощью алгоритма Word2Vec с позиционным окном в качестве контекста. Тогда мы с помощью матрицы $\boldsymbol{E}$ будем отображать слова на векторы погружения $\boldsymbol{e}_i = \boldsymbol{E}_{[w_i]}$. Этот метод неплохо работает, но ему может мешать недостаток покрытия, в случае когда словарные элементы не встречались на этапе обучения или предобучения. Слова состоят из литер, и некоторые суффиксы и префиксы, а также другие орфографические признаки, такие как заглавная буква в начале, наличие дефисов или цифр, могут дать весомые подсказки относительно разрешения неоднозначности слова. В главах 7 и 8 мы обсуждали, как включить такую информацию с помощью специальных признаков. Сейчас мы заменим такие вручную спроектированные экстракторы признаков на РНС. Точнее, мы будем использовать две РНС уровня литер. Для слова w, состоящего из литер $c_1, \ldots, c_\ell$, отобразим каждую литеру в соответствующий вектор погружения $\boldsymbol{c}_i$. Затем будем кодировать слово, применяя прямую и обратную РНС по литерам. Эти РНС смогут либо заменить вектор погружения слова, либо – что еще лучше – произвести конкатенацию с ним:

$$\boldsymbol{x}_i = \phi(s, i) = [\boldsymbol{E}_{[w_i]}; \mathrm{RNN}^{\mathrm{f}}(\boldsymbol{c}_{1:\ell}); \mathrm{RNN}^{\mathrm{b}}(\boldsymbol{c}_{\ell:1})].$$

Отметим, что прямая РНС занимается улавливанием суффиксов, а обратная – префиксов и что обе РНС можно сделать чувствительными к заглавным буквам, дефисам и даже длине слова.

ОКОНЧАТЕЛЬНАЯ МОДЕЛЬ Таким образом, модель разметки принимает вид:

$$\begin{aligned} p(t_i = j | w_1, \ldots, w_n) &= \mathrm{softmax}(\mathrm{MLP}(\mathrm{biRNN}(\boldsymbol{x}_{1:n}, i)))_{[j]}; \\ \boldsymbol{x}_i &= \phi(s, i) = [\boldsymbol{E}_{[w_i]}; \mathrm{RNN}^{\mathrm{f}}(\boldsymbol{c}_{1:\ell}); \mathrm{RNN}^{\mathrm{b}}(\boldsymbol{c}_{\ell:1})]. \end{aligned} \tag{16.4}$$

В качестве функции потерь используется перекрестная энтропия. Прореживание слов (раздел 8.4.2) для погружений слов дает положительный эффект. Архитектура показана на рис. 16.1.

Похожая модель разметки описана в работе Plank et al. [2016], где видно, что она дает вполне конкурентоспособные результаты для широкого круга языков.

СВЕРТКА И ПУЛИНГ НА УРОВНЕ ЛИТЕР В описанной выше архитектуре слова отображаются на векторы с помощью прямой и обратной РНС по литерам слов. Альтернатива – представлять слова с помощью сверточно-пулинговых нейронных сетей на уровне литер (*см.* главу 13). В работе Ma and Hovy [2016] продемонстрировано, что применение одного сверточно-пулингового слоя с размером окна $k = 3$ к литерам слова действительно эффективно в задачах частеречной разметки и распознавания именованных сущностей.

СТРУКТУРНЫЕ МОДЕЛИ В модели выше предсказание метки i-го слова производится независимо от других меток. Это может хорошо работать, но можно было бы обусловить i-ю метку предыдущими предсказаниями модели. В качестве условия можно брать либо k предыдущих меток (в согласии с марковским предположением), и тогда используются погружения меток $\boldsymbol{E}_{[t]}$, что дает:

$$p(t_i = j|w_1, \ldots, w_n, t_{i-1}, \ldots, t_{i-k}) = \text{softmax}(\text{MLP}([\text{biRNN}(\boldsymbol{x}_{1:n}, i); \boldsymbol{E}_{[t_{i-1}]}; \ldots; \boldsymbol{E}_{[t_{i-k}]}]))_{[j]},$$

либо всю последовательность предыдущих предсказаний $t_{1:i-1}$, и тогда для кодирования последовательности меток применяется РНС:

$$p(t_i = j|w_1, \ldots, w_n, t_{1:i-1}) = \text{softmax}(\text{MLP}([\text{biRNN}(\boldsymbol{x}_{1:n}, i); \text{RNN}^t(\boldsymbol{t}_{1:i-1})]))_{[j]},$$

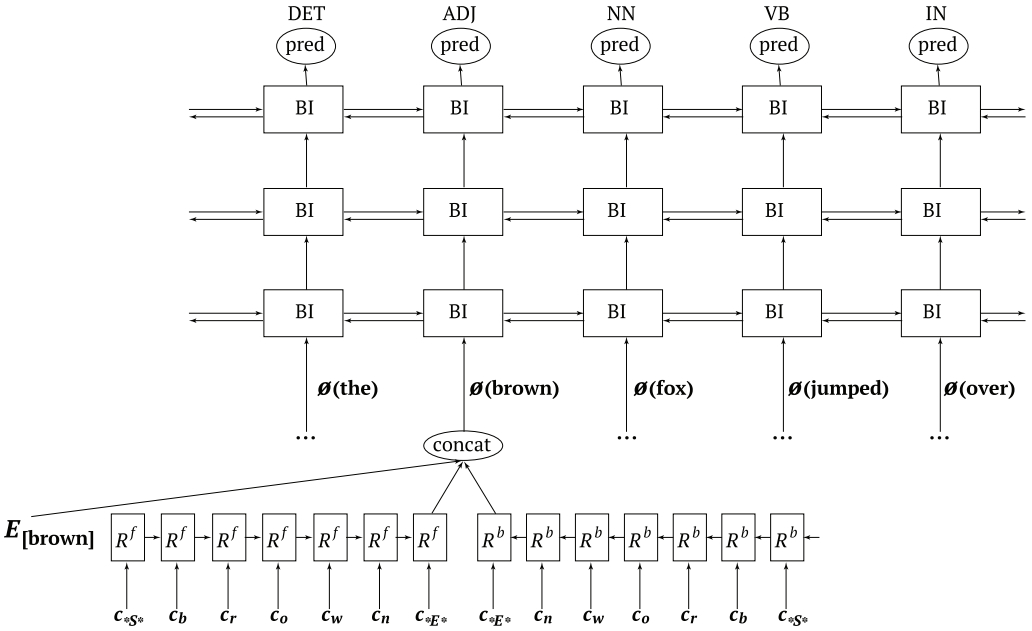


Рис. 16.1 ❖ Архитектура разметки с применением РНС. Каждое слово w_i преобразуется в вектор $\phi(w_i)$, являющийся конкатенацией вектора погружения и конечных состояний прямой и обратной РНС на уровне литер. Затем векторы слов подаются на вход глубокой biRNN. Выходное состояние каждой biRNN внешнего слоя подается на вход предсказывающей сети (МСП в сочетании с softmax), которая дает предсказание метки. Отметим, что предсказание каждой метки может быть обусловлено всей входной последовательностью

В обоих случаях модель можно выполнять в жадном режиме, когда метки t_i предсказываются последовательно, либо в режиме динамического программирования (в марковском случае) или лучевого поиска (в обоих случаях) для нахождения последовательности меток с наивысшей оценкой. Такая модель использовалась для CCG-суперразметки (когда каждому слову назначается одна из большого множества меток, кодирующих развитую синтаксическую структуру) в работе Vaswani et al. [2016]. Обучение структурному предсказанию для таких моделей обсуждается в главе 19.

16.2.2. Классификация документов с применением РНС-СНС

В примерах классификации по эмоциональной окраске из раздела 16.1.1 мы подавали векторы погружения на входы прямой и обратной РНС, за которыми располагался слой классификации [уравнения (16.2)]. В примере разметчика из раздела 16.2.1 мы видели, что погружения слов можно дополнить (или заменить) моделями уровня литер, например литерными РНС или СНС, чтобы улучшить покрытие модели и помочь ей справляться со словами, словоизменениями и опечатками, которых она не видела на этапе обучения.

Такой же подход может оказаться эффективен для классификации документов, только вместо подачи погружений слов на входы двух РНС мы подаем векторы, получившиеся применением литерной РНС или сверточно-пулингового слоя к каждому слову.

Еще одна альтернатива – применить иерархическую сверточно-пулинговую сеть (раздел 13.3) к литерам с целью получить более короткую последовательность векторов, представляющих блоки, большие, чем литеры, но необязательно слова (уловленная информация может относиться к образованию как большему, так и меньшему одного слова), а затем подать эту последовательность двум РНС и слою классификации. Такой подход был исследован в работе Xiao and Cho [2016] в применении к нескольким задачам классификации документов. Точнее, предложенная ими иерархическая архитектура включала ряд сверточных и пулинговых слоев. В каждом слое к последовательности входных векторов применялась свертка с окном размера k, а затем к каждым двум соседним результирующим векторам применялся max-пулинг, что сокращало размер последовательности вдвое. После нескольких таких слоев (с окнами размера от 5 до 3, зависящего от номера слоя, например 5, 5, 3) получившиеся векторы подавались на вход прямой и обратной РНС типа GRU, а их результаты – на вход компонента классификации (полносвязный слой с функцией активации softmax). Кроме того, между последним сверточным слоем и РНС, а также между РНС и компонентом классификации применялось прореживание. Этот подход доказал свою эффективность в нескольких задачах классификации документов.

16.2.3. Анализ зависимостей методом разложения на дуги

Вернемся к задаче анализа зависимостей методом разложения на дуги из раздела 7.7. Напомним, что нам дано предложение *sent*, состоящее из слов $w_{1:n}$, и соответствующие метки частей речи $t_{1:n}$, а требуется назначить каждой паре (w_i, w_j) оценку, описывающую вероятность того, что слово w_i является заглавным для w_j. В разделе 8.6 мы вывели мудреную функцию выделения признаков для этой задачи, основанную на окнах, окружающих заглавные слова и модификаторы, словах между заглавным и модификаторами и их метках частей речи. Эту сложную функцию можно заменить конкатенацией векторов двух biRNN, соответствующих заглавному слову и модификатору.

Точнее, пусть даны слова и метки частей речи $w_{1:n}$ и $t_{1:n}$ и соответствующие им векторы погружений $\boldsymbol{w}_{1:n}$ и $\boldsymbol{t}_{1:n}$. Создадим biRNN, кодирующую $\boldsymbol{v}_i$ для каждой позиции предложения, для чего конкатенируем векторы слов и меток частей речи и подадим их на вход глубокой biRNN:

$$\begin{aligned} \boldsymbol{v}_{1:n} &= \text{biRNN}^*(\boldsymbol{x}_{1:n}); \\ \boldsymbol{x}_i &= [\boldsymbol{w}_i; \boldsymbol{t}_i]. \end{aligned} \tag{16.5}$$

Затем оценим пару заглавное слово–модификатор, пропустив конкатенацию векторов, выданных biRNN, через МСП:

$$\text{ArcScore}(h, m, w_{1:n}, t_{1:n}) = \text{МСП}(\phi(h, m, s)) = \text{МСП}([\boldsymbol{v}_h; \boldsymbol{v}_m]). \tag{16.6}$$

Эта архитектура показана на рис. 16.2. Заметим, что выдаваемые biRNN векторы $\boldsymbol{v}_i$ кодируют слова в контексте, т. е. по сути дела формируют двустороннее бесконечное окно вокруг слов w_i, чувствительное как к последовательности меток частей речи, так и к последовательности слов. Кроме того, конкатенация $[\boldsymbol{v}_h; \boldsymbol{v}_m]$ включает результаты работы РНС до каждого слова включительно в обоих направлениях и, в частности, покрывает последовательность позиций между словами w_h и w_m. biRNN обучается в составе большей сети с целью уловить аспекты последовательности, важные для задачи синтаксического анализа (структурное обучение анализатора, применяющего метод разложения на дуги, объясняется в разделе 19.4.1).

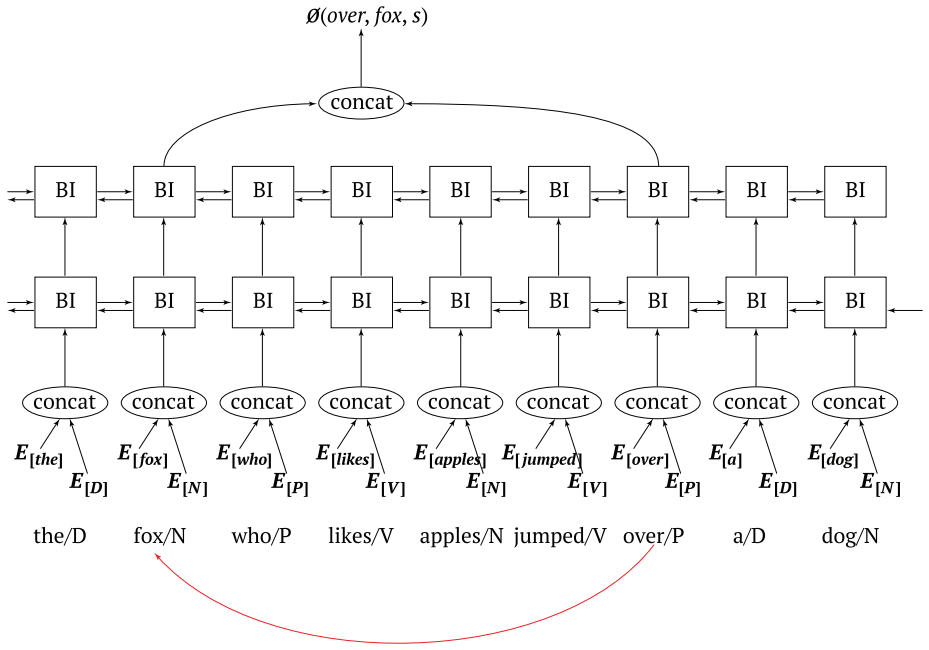


Рис. 16.2 ❖ Иллюстрация экстрактора признаков для анализатора методом разложения на дуги для дуги между словами *fox* и *over*

Такой экстрактор признаков использовался в работе Kiperwasser and Goldberg [2016b], где продемонстрировал лучшие в отрасли результаты синтаксического анализа методом разложения на дуги, вполне сопоставимые с результатами го-

раздо более сложных моделей. Похожий подход был принят в работе Zhang et al. [2016] и показал аналогичные результаты в другом режиме обучения.

Вообще, всякий раз, как слова используются в качестве признаков в задаче, чувствительной к порядку слов или структуре предложения, слова можно заменять соответствующими им векторами, обученными сетью biLSTM. Такой подход применялся в работах Kiperwasser and Goldberg [2016b] и Cross and Huang [2016a,b] в контексте синтаксического разбора *на основе перехода состояний* и показал впечатляющие результаты.

Глава 17

Условная генерация

В главе 14 было отмечено, что РНС могут работать как немарковские языковые модели, обусловленные всей историей. Эта особенность делает их пригодными для использования в качестве *генераторов* (порождающих последовательности слов на естественном языке) и *условных генераторов*, для которых сгенерированный выход обусловлен сложным входом. В этой главе обсуждаются такие архитектуры.

17.1. РНС-генераторы

Частным случаем применения архитектуры РНС-преобразователя для языкового моделирования (раздел 14.3.3) является *генерация последовательностей*. Для генерации можно использовать любую языковую модель из описанных в разделе 9.5. В случае РНС-преобразователя генератор связывает выход преобразователя в момент i с его входом в момент $i + 1$: после предсказания распределения следующих выходных символов $p(ti = k|t_{1:i-1})$ выбирается лексема t_i, и соответствующий ей вектор погружения подается на вход следующего шага. Процесс останавливается после генерации специального символа конца последовательности, который часто обозначается </s>. Это процесс показан на рис. 17.1.

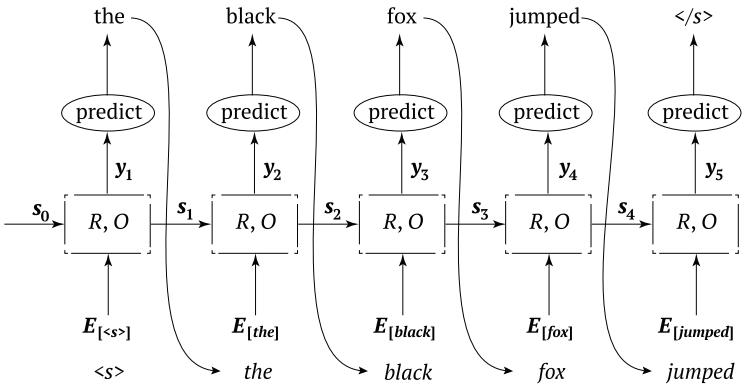


Рис. 17.1 ❖ РНС-преобразователь в роли генератора

Как и в случае генерации по n-граммной языковой модели (раздел 9.5), при генерации с помощью обученной РНС-преобразователя можно либо брать на каждом шаге элемент с наибольшей вероятностью, либо случайно выбирать элемент из предсказанного моделью распределения, либо использовать лучевой поиск для нахождения элемента с глобально наибольшей вероятностью.

Впечатляющую демонстрацию способности вентильных РНС использовать в качестве условия произвольно длинную историю дает основанная на РНС языковая модель, обученная на литерах, а не на словах. При использовании в роли генератора перед моделью ставится задача порождать случайные предложения литера за литерой, причем каждая литера обусловлена предыдущими [Sutskever et al., 2011]. Работа на уровне литер заставляет модель заглядывать дальше в прошлое, чтобы связать буквы в слова, а слова в предложения, образуя осмысленные комбинации. Сгенерированные тексты не только напоминают текст на приличном английском языке, но и демонстрируют чувствительность к таким свойствам, которые не улавливаются n-граммными языковыми моделями, в частности к длине строк и сбалансированности вложенных скобок. При обучении на исходных кодах на языке C сгенерированные последовательности соблюдали стиль отступов и общие синтаксические ограничения языка. Интересную демонстрацию и анализ свойств языковых моделей литерного уровня на основе РНС *см*. в работе Karpathy et al. [2015].

17.1.1. Обучение генераторов

Общий подход к обучению генератора заключается в том, чтобы обучать его как преобразователь, перед которым стоит цель сосредоточить большую массу вероятности на следующей лексеме в наблюдаемой последовательности, исходя из ранее наблюдавшихся лексем (т. е. обучить языковую модель).

Точнее, для каждого предложения из n слов $w_1, \ldots, w_n$ в обучающем корпусе мы порождаем РНС-преобразователь с $n + 1$ входами и $n + 1$ соответствующими выходами, где первым входом является символ начала последовательности, а за ним следуют n слов предложения. Тогда первым ожидаемым выходом будет w_1, вторым – w_2 и т. д., а $(n + 1)$-м ожидаемым выходом будет символ конца последовательности.

Такой подход к обучению часто называют *принуждением со стороны учителя* (teacher-forcing), поскольку на вход генератору подается наблюдаемое слово, даже если его собственный предиктор сопоставляет ему небольшую массу вероятности и на этапе тестирования он сгенерировал бы в этом состоянии другое слово.

Хотя эта идея работает, она плохо справляется с отклонениями от золотых последовательностей. Действительно, если на вход генератору подавать его собственные предсказания, а не золотые последовательности, то он должен будет назначать вероятности при условии состояний, не наблюдавшихся во время обучения. Лучевой поиск высоковероятной выходной последовательности также мог бы выиграть от специализированной процедуры обучения. На момент написания книги вопрос о том, как справляться с такими ситуациями, остается открытым и выходит за рамки книги. Вкратце мы коснемся его, когда будет обсуждать структурные предсказания в разделе 19.3.

17.2. Условная генерация (кодировщик–декодер)

Хотя использование РНС в роли генератора – эффектная демонстрация ее возможностей, по-настоящему мощь РНС-преобразователя раскрывается при переходе к системе *условной генерации*.

Система генерации порождает следующую лексему t_{j+1} на основе ранее сгенерированных лексем $\hat{t}_{1:j}$:

$$\hat{t}_{j+1} \sim p(t_{j+1} = k | \hat{t}_{1:j}). \tag{17.1}$$

С помощью РНС это моделируется так:

$$\begin{aligned} p(t_{j+1} = k | \hat{t}_{1:j}) &= f(\text{RNN}(\hat{\boldsymbol{t}}_{1:j})); \\ \hat{t}_j &\sim p(t_j = k | \hat{t}_{1:j-1}) \end{aligned} \tag{17.2}$$

или, если расписать рекурсивное определение более детально:

$$\begin{aligned} p(t_{j+1} = k | \hat{t}_{1:j}) &= f(O(\boldsymbol{s}_{j+1})); \\ \boldsymbol{s}_{j+1} &= R(\hat{\boldsymbol{t}}_j, \boldsymbol{s}_j); \\ \hat{t}_j &\sim p(t_j = k | \hat{t}_{1:j-1}), \end{aligned} \tag{17.3}$$

где f – параметрическая функция, отображающая состояние РНС в распределение слов, например $f(x) = \text{softmax}(\boldsymbol{x}\boldsymbol{W} + \boldsymbol{b})$ или $f(x) = \text{softmax}(\text{MLP}(\boldsymbol{x}))$.

В системе условной генерации следующая лексема порождается на основе ранее сгенерированных лексем и дополнительного обусловливающего контекста c:

$$\hat{t}_{1:j-1} \sim p(t_{j+1} = k | \hat{t}_{1:j}, c). \tag{17.4}$$

При использовании РНС контекст c представляется вектором $\boldsymbol{c}$:

$$\begin{aligned} p(t_{j+1} = k | \hat{t}_{1:j}, c) &= f(\text{RNN}(\boldsymbol{v}_{1:j})); \\ \boldsymbol{v}_i &= [\hat{\boldsymbol{t}}_i; \boldsymbol{c}]; \\ \hat{t}_j &\sim p(t_j | \hat{t}_{1:j-1}, c) \end{aligned} \tag{17.5}$$

или, применяя рекурсивное определение:

$$\begin{aligned} p(t_{j+1} = k | \hat{t}_{1:j}, c) &= f(O(\boldsymbol{s}_{j+1})); \\ \boldsymbol{s}_{j+1} &= R(\boldsymbol{s}_j, [\hat{\boldsymbol{t}}_j; \boldsymbol{c}]); \\ \hat{t}_j &\sim p(t_j | \hat{t}_{1:j-1}, c). \end{aligned} \tag{17.6}$$

На каждом этапе процесса генерации контекстный вектор $\boldsymbol{c}$ конкатенируется с входом $\hat{\boldsymbol{t}}_j$, и результат подается на вход РНС, что приводит к следующему предсказанию. Эта архитектура показана на рис. 17.2.

Какого рода информацию можно закодировать в контексте c? Да практически все данные, которые доступны во время обучения и сочтены полезными. Например, если имеется большой корпус новых объектов, распределенных по различным темам, мы можем считать тему обусловливающим контекстом. Тогда наша языковая модель сможет генерировать тексты на заданную тему. Если нас интересуют рецензии на фильмы, то можно обусловить генерацию жанром фильма, рейтингом рецензии и, быть может, географическим местонахождением автора.

Затем эти аспекты можно будет контролировать в процессе генерации текста. Можно также использовать в качестве условий *выведенные* свойства, автоматически выделяемые из текста. Например, в качестве такого свойства можно взять эвристики, которые сообщают, написано ли данное предложение от первого лица, содержит ли оно конструкции в страдательном залоге, а также уровень использованного словаря. Впоследствии эти аспекты можно будет использовать как обусловливающий контекст на этапе обучения и далее для генерации текста.

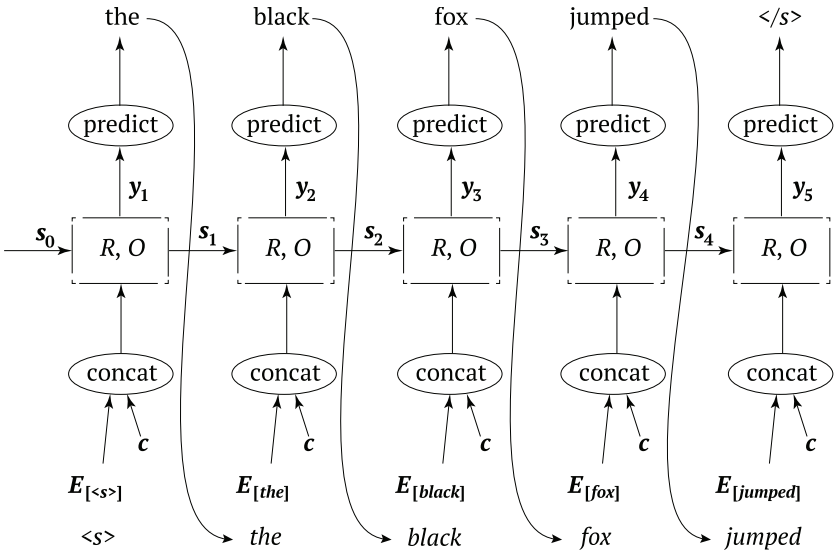


Рис. 17.2 ❖ Условный РНС-генератор

17.2.1. Модели типа последовательность-в-последовательность

Контекст c может принимать много форм. В предыдущем разделе мы описали некоторые примеры обусловливающих контекстов фиксированной длины, напоминающих множества. Другой популярный подход – взять в качестве c другую последовательность, чаще всего фрагмент текста. Это открывает путь к системе условной генерации типа *последовательность-в-последовательность*, которая также называется системой *кодировщик–декодер* [Cho et al., 2014a, Sutskever et al., 2014].

В условной генерации последовательность-в-последовательность дана исходная последовательность $\boldsymbol{x}_{1:n}$ (например, предложение на французском языке), а требуется сгенерировать выходную последовательность $\boldsymbol{t}_{1:m}$ (например, его перевод на английский язык). Для этого мы должны *закодировать* исходное предложение $\boldsymbol{x}_{1:n}$ в вектор с помощью функции кодирования $\boldsymbol{c} = \text{Enc}(\boldsymbol{x}_{1:n})$, которая обычно реализована в виде РНС: $\boldsymbol{c} = \text{RNN}^{\text{enc}}(\boldsymbol{x}_{1:n})$. Затем используется РНС, реализующая условный генератор (*декодер*), которая генерирует требуемый выход $\boldsymbol{t}_{1:m}$ в соответствии с уравнениями (17.5). Эта архитектура показана на рис. 17.3.

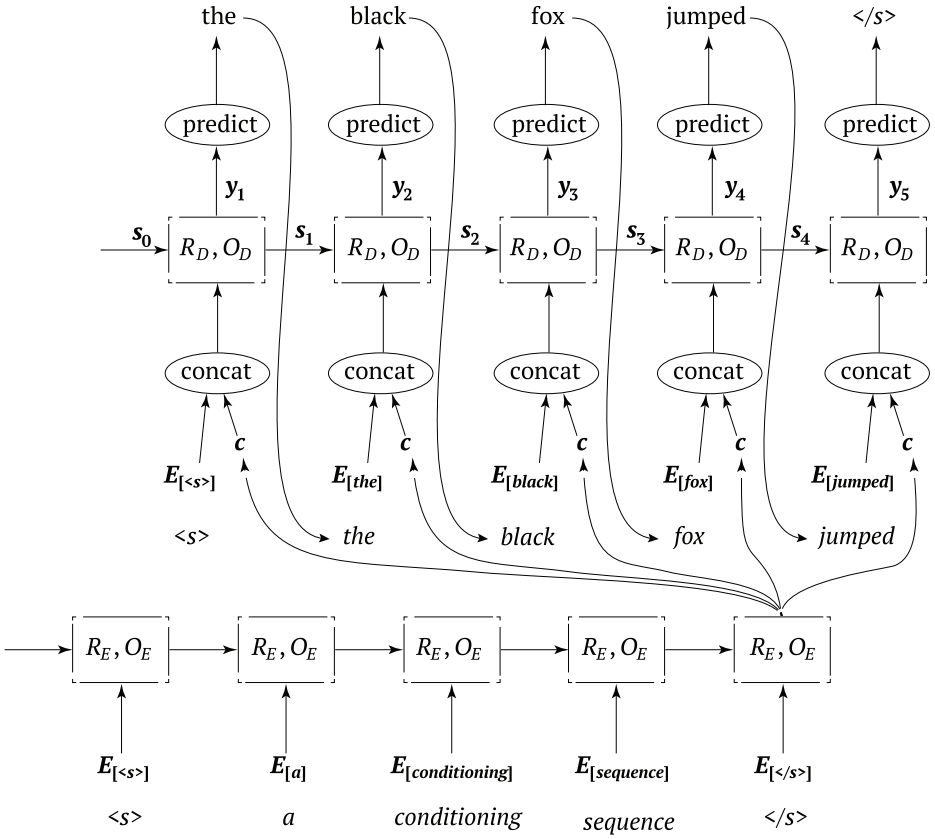


Рис. 17.3 ❖ РНС в роли генератора последовательность-в-последовательность

Такая конфигурация полезна для отображения последовательностей длины n в последовательности длины m. Кодировщик преобразует исходное предложение в вектор $\boldsymbol{c}$, а затем декодер предсказывает (с помощью целевой функции языкового моделирования) конечную последовательность слов, обусловленную ранее предсказанными словами, а также закодированным предложением $\boldsymbol{c}$. РНС, исполняющие роли кодировщика и декодера, обучаются совместно. Вмешательство учителя необходимо только для декодирующей РНС, но градиенты распространяются и в кодирующую РНС (рис. 17.4).

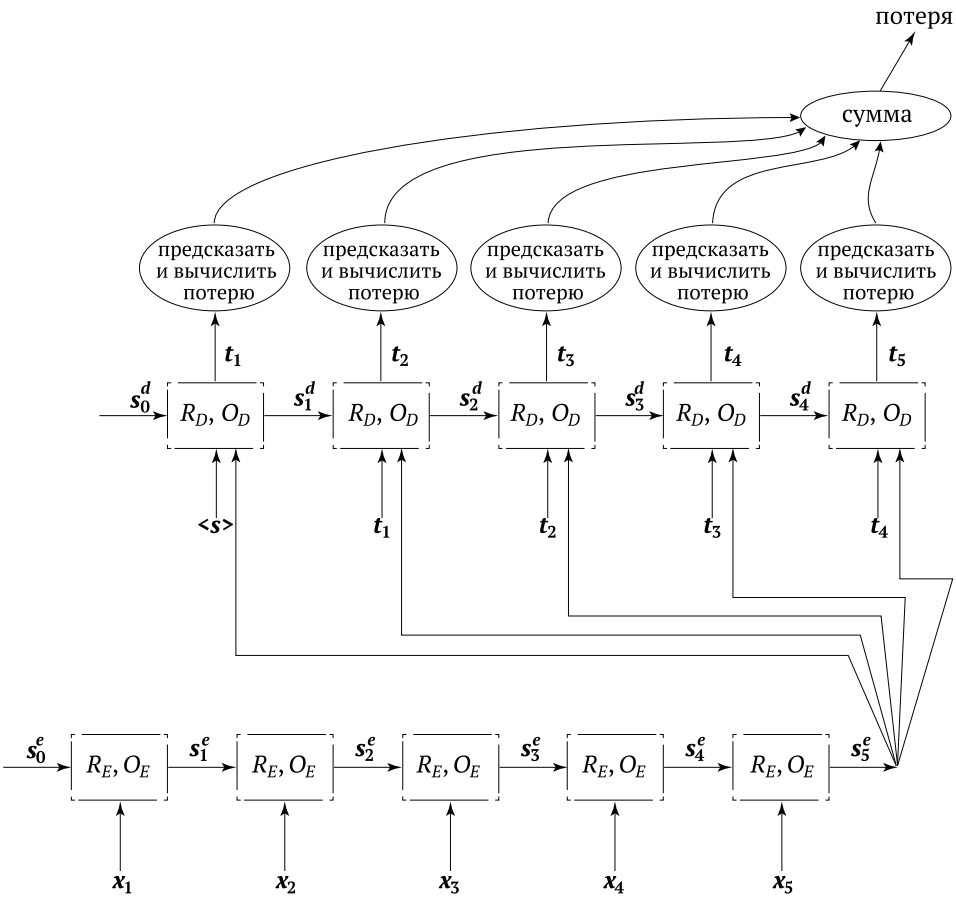


Рис. 17.4 ❖ Граф обучения РНС типа последовательность-в-последовательность

17.2.2. Приложения

РНС типа последовательность-в-последовательность – очень общий подход, который может пригодиться в любой ситуации, когда требуется отображать входную последовательность в выходную. Перечислим несколько примеров, описанных в литературе.

МАШИННЫЙ ПЕРЕВОД Подход последовательность-в-последовательность оказался на удивление эффективен в машинном переводе [Sutskever et al., 2014], где использовались глубокие РНС типа LSTM. Как показали авторы, этот метод особенно хорошо работает, если вводить исходное предложение «задом наперед», так что $\boldsymbol{x}_n$ соответствует первому слову. Так, второй РНС легче установить связь между первыми словами исходного и конечного предложений.

Хотя успех этого подхода в применении к переводу с французского на английский впечатляет, стоит отметить, что авторам работы Sutskever et al. [2014] по-

требовалось восемь слоев LSTM высокой размерности, поэтому с вычислительной точки зрения сеть обходится дорого, а ее обучение – нетривиальная задача. Ниже в этой главе мы опишем *архитектуры на основе внимания* – развитие архитектуры последовательность-в-последовательность, гораздо более полезное для машинного перевода.

АВТОМАТИЧЕСКИЙ ОТВЕТ НА ПИСЬМА В этой задаче требуется отобразить потенциально длинное сообщение электронной почты в короткий ответ типа *Да*, *Сделаю*, *Отлично, В среду увидимся* или *Так не пойдет*. В работе Kannan et al. [2016] описана реализация функции автоответа в программе Google Inbox. В основе решения лежит прямолинейная модель условной генерации типа последовательность-в-последовательность, в которой LSTM-кодировщик читает письма, а LSTM-декодер генерирует ответы. Этот компонент был обучен на большом числе пар письмо–ответ. Конечно, чтобы успешно интегрировать такой компонент с продуктом, необходимы дополнительные модули, которые следят за вызовом компонента, обеспечивают разнообразие ответов и баланс между положительными и отрицательными ответами, гарантируют конфиденциальность данных пользователя и т. д. Детали *см*. в работе Kannan et al. [2016].

МОРФОЛОГИЧЕСКОЕ СЛОВОИЗМЕНЕНИЕ В этой задаче входом является базовое слово и требуемый вид словоизменения, а выходом – измененная форма слова. Например, если задано финское слово *bruttoarvo* и требуемое словоизменение pos = N, case = IN + ABL, num = PL, то выходом будет *bruttoarvoista*. Традиционно эта задача решалась с помощью составленных вручную лексиконов и преобразователей в виде конечных автоматов, но она также прекрасно подходит для применения моделей условной генерации последовательность-в-последовательность на уровне литер [Faruqui et al., 2016]. Результаты конкурса SIGMORPHON 2016 по генерации словоизменений показывают, что подходы на основе рекуррентных нейронных сетей превосходят все остальные [Cotterell et al., 2016]. В системе, занявшей второе место [Aharoni et al., 2016], использовалась модель последовательность-в-последовательность с несколькими предметно-ориентированными усовершенствованиями, а в победившей системе [Kann and Schütze, 2016] – ансамбль *аттентивных* моделей последовательность-в-последовательность типа описанной в разделе 17.4.

ПРОЧИЕ ПРИМЕНЕНИЯ Отображение последовательности n элементов в последовательность m элементов носит очень общий характер, так что почти любую задачу можно сформулировать как сочетание кодирования и генерации. Однако из того, что задачу *можно* сформулировать таким образом, не следует, что так и следует поступить – быть может, другие архитектуры лучше подходят для нее или их проще обучить. Далее мы опишем несколько приложений, которые трудно – да и не нужно – обучать в системе кодировщик–декодер, но для которых существуют другие, лучше отвечающие задаче архитектуры. Тот факт, что авторам удалось получить приемлемую верность с использованием кодировщика и декодера, – свидетельство в пользу мощности этого подхода.

В работе Filippova et al. [2015] эта архитектура используется для *сжатия предложения посредством удаления*. В этой задаче дано предложение, например «*Alan Turing, known as the father of computer science, the codebreaker that helped win World War 2, and the man tortured by the state for being gay, is to receive a pardon nearly 60 years*

after his death»[1], и требуется породить его более короткий («сжатый»), но сохранивший основную информацию вариант путем удаления некоторых слов. Примером сжатия может служить «*Alan Turing is to receive a pardon*». В указанной работе проблема моделируется как отображение последовательность-в-последовательность, в котором входной последовательностью является входное предложение (возможно, в сочетании с синтаксической информацией, выведенной из автоматического порожденного дерева разбора), а выходной – последовательность решений Keep (оставить), Delete (удалить) и Stop (остановить). Модель была обучена на корпусе, содержащем примерно 2 000 000 пар предложение – сжатый вариант, автоматически выделенных из новостей [Filippova and Altun, 2013], и дала результаты, не уступающие лучшим в отрасли[2].

В работе Gillick et al. [2016] задача частеречной разметки и распознавания именованных сущностей рассматривается как проблема отображения последовательности байтов в кодировке Юникод в последовательность интервальных предсказаний вида S12,L13,PER,S40,L11,LOC, означающего, что 13-байтовая сущность Person начинается в позиции 12, а 11-байтовая сущность Location – в позиции 40[3].

В работе Vinyals et al. [2014] синтаксический разбор реализован как задача отображения предложения во множество решений о построении дерева составляющих.

17.2.3. Другие обусловливающие контексты

Условная генерация – очень гибкое решение: кодировщик не обязан быть РНС. На самом деле вектор обусловливающего контекста может быть основан на одном слове или на кодировании в виде CBOW, он может порождаться сверточной сетью или в результате еще какого-то сложного вычисления.

Обуславливающий контекст даже не обязан быть текстовым. В задаче о диалоге (когда РНС обучается порождать ответы на сообщения в диалоге) в работе Li et al. [2016] в качестве контекста использован обучаемый вектор погружения, ассоциированный с *пользователем*, который написал ответ. Интуитивно понятно, что у разных пользователей разный стиль общения, зависящий от возраста, пола, социальной роли, образования, черт характера и многих других скрытых факторов. Получая в качестве условия пользователя, давшего ответ, сеть может обучиться

---

[1] Алан Тьюринг, известный как отец информатики, криптоаналитик, который помог выиграть Вторую мировую войну, и человек, преследуемый государством за гомосексуальность, должен получить помилование спустя почти 60 лет после смерти. – *Прим. перев.*

[2] Хотя результаты впечатляют, подход последовательность-в-последовательность, пожалуй, избыточен для данной задачи, где требуется отобразить последовательность n слов в последовательность n решений, в которой i-е решение точно соответствует i-му слову. По существу, это задача о разметке последовательности, поэтому для нее больше подошел бы biLTSM-преобразователь типа описанных в предыдущей главе. На самом деле в работе Klerke et al. [2016] показано, что сравнимой (разве что чуть меньшей) верности позволяет добиться biRNN-преобразователь, для обучения которого потребовалось *на несколько порядков меньше* данных.

[3] И это тоже задача о разметке последовательностей, которую лучше решать с помощью biLSTM-преобразователя или структурного biLSTM-преобразователя (biLSTM-CRF), описанного в разделе 19.4.2.

адаптироваться к его предсказаниям, не отказываясь от базовой языковой модели, взятой за основу. Более того, в качестве побочного эффекта обучения генератора сеть обучается еще и *погружениям пользователей*, т. е. порождает похожие векторы для пользователей с похожим стилем общения. На этапе тестирования можно влиять на стиль сгенерированного ответа, подавая на вход конкретного пользователя (или усредненный вектор пользователей) в качестве обусловливающего контекста.

Отходя еще дальше от лингвистических проблем, упомянем популярную задачу об *описании изображений*: входное изображение кодируется вектором (обычно с помощью многослойной сверточной сети[1]), и этот вектор используется в качестве обусловливающего контекста для РНС-генератора, который обучается предсказывать сопроводительные подписи к изображениям [Karpathy and Li, 2015, Mao et al., 2014, Vinyals et al., 2015].

В работе Huang et al. [2016] задача описания обобщена до более амбициозной задачи *визуального повествования* (visual story telling), когда на вход подается последовательность изображений, а на выходе получается рассказ, описывающий смену изображений. В данном случае кодировщиком является РНС, читающая последовательность векторов изображений.

17.3. Установление сходства предложений без учителя

Часто желательно иметь такое векторное представление предложений, при котором похожим изображениям соответствуют похожие векторы. Эта задача не очень корректно поставлена (что значит *похожие предложения*?) и пока остается открытым направлением исследований, но есть подходы, дающие вполне разумные результаты. Здесь мы ограничимся обучением без учителя, когда сеть обучается на неаннотированных данных. Результатом обучения является *кодировщик*, т. е. функция $\mathrm{Enc}(w_{1:n})$, которая преобразует похожие предложения в похожие векторы.

Большинство подходов основано на системе последовательность-в-последовательность: кодирующая РНС обучается порождать контекстные векторы $\boldsymbol{c}$, которые используются декодирующей РНС для решения задачи. Следовательно, $\boldsymbol{c}$ должен улавливать информацию из предложения, важную для задачи. Затем декодирующая РНС отбрасывается, а кодировщик используется для генерации представлений предложений $\boldsymbol{c}$ в предположении, что у похожих предложений будут похожие векторы. Результирующая функция сходства при этом принципиально зависит от задачи, для которой обучался декодер.

АВТОКОДИРОВКА Подход на основе автокодировки – это модель условной генерации, в которой предложение кодируется с помощью РНС, а затем декодер пытается реконструировать входное предложение. Таким образом, модель обучается кодировать информацию, необходимую для реконструкции, опять-таки в надеж-

[1] Отображение изображений в векторы с помощью нейросетевых архитектур – хорошо изученное направление со сложившейся практикой и многочисленными достижениями. Но это выходит за рамки книги.

де, что у похожих последовательностей будут похожие векторы. Впрочем, целевая функция реконструкции предложений, пожалуй, не идеальна для общей задачи об установлении сходства предложений, поскольку она с большой вероятностью дает сильно различающиеся представления предложений с одинаковым смыслом, но разными словами.

МАШИННЫЙ ПЕРЕВОД В этом случае сеть, преобразующая последовательность в последовательность, обучается переводить предложения с английского на какой-то другой язык. Интуитивно кажется, что векторы, порожденные кодировщиком, полезны для перевода, поскольку они правильно кодируют суть переводимого предложения и, следовательно, предложениям с похожим переводом будут соответствовать похожие векторы. Для обучения сети условной генерации в этом методе необходим большой корпус параллельных текстов, используемых в машинном переводе.

SKIP-THOUGHTS Модель из работы Kiros et al. [2015], названная авторами *векторами skip-thought*, предлагает интересную целевую функцию для задачи о сходстве предложений. Модель обобщает дистрибутивную гипотезу со слов на предложения, утверждая, что предложения похожи, если встречаются в похожих контекстах, где под текстом предложения понимаются окружающие его предложения. Таким образом, модель skip-thoughts – это модель условной генерации, в которой кодирующая RNC отображает предложение в вектор, а затем декодер обучается реконструировать предложение по его представлению. Обученный методом skip-thoughts кодировщик на практике дает впечатляющие результаты, отображая предложения вида

(a) *he ran his hand inside his coat, double-checking that the unopened letter was still there*[1] и

(b) *he slipped his hand between his coat and his shirt, where the folded copies lay in a brown envelope*[2]

в похожие векторы.

СИНТАКСИЧЕСКОЕ СХОДСТВО В работе Vinyals et al. [2014] продемонстрирована пара кодировщик–декодер, способная давать достойные результаты для *фразового синтаксического разбора* (phrase-based syntactic parsing): предложение кодируется так, чтобы декодер мог реконструировать дерево разбора, линеаризованное в виде потока заключенных в скобки принятых решений, т. е. предложение

the boy opened the door (мальчик открыл дверь)

представляется в виде

(S (NP DT NN) (VP VBD (NP DT NN))).

Представления, полученные при таком обучении, могут улавливать синтаксическую структуру предложения.

---

[1] Он сунул руку под пиджак, еще раз проверив, что нераспечатанное письмо все еще там. – *Прим. перев.*

[2] Он провел рукой между пиджаком и рубашкой, где в коричневом конверте лежали сложенные листки. – *Прим. перев.*

17.4. Условная генерация с вниманием

В сетях типа кодировщик–декодер, описанных в разделе 17.2, входное предложение кодируется одним вектором, который затем используется в качестве обусловливающего контекста для RHC-генератора. Эта архитектура понуждает закодированный вектор $\boldsymbol{c} = \text{RNN}^{\text{enc}}(\boldsymbol{x}_{1:n})$ включать всю информацию, необходимую для генерации, а от генератора требуется извлекать эту информацию из вектора фиксированной длины. Удивительно, что при таких сильных ограничениях архитектура дает прекрасные результаты. Но во многих случаях ее можно значительно улучшить, добавив *механизм внимания*. *Архитектура условной генерации с вниманием* [Bahdanau et al., 2014] ослабляет условие, согласно которому все исходное предложение должно быть закодировано одним вектором. Вместо этого предложение кодируется последовательностью векторов, а декодер применяет *механизм мягкого внимания*, чтобы решить, каким частям закодированного входа уделить особое внимание. Кодировщик, декодер и механизм внимания обучаются совместно, чтобы они могли работать сообща.

Точнее, архитектура кодировщик–декодер с вниманием кодирует входную последовательность длины n $\boldsymbol{x}_{1:n}$ с помощью сети biRNN, порождая n векторов $\boldsymbol{c}_{1:n}$:

$$\boldsymbol{c}_{1:n} = \text{Enc}(\boldsymbol{x}_{1:n}) = \text{biRNN}^{*}(\boldsymbol{x}_{1:n}).$$

Генератор (декодер) может затем использовать эти векторы как постоянную память, представляющую обусловливающее предложение: на любом этапе j процесса генерации он выбирает, какому из векторов $\boldsymbol{c}_{1:n}$ уделить внимание, т. е. определяет фокусный контекстный вектор $\boldsymbol{c}^j = \text{attend}(\boldsymbol{c}_{1:n}, \hat{t}_{1:j})$.

Затем этот вектор $\boldsymbol{c}^j$ используется для обусловливания генерации на шаге j:

$$\begin{aligned} p(t_{j+1} = k | \hat{t}_{1:j}; \boldsymbol{x}_{1:n}) &= f(O(\boldsymbol{s}_{j+1})); \\ \boldsymbol{s}_{j+1} &= R(\boldsymbol{s}_j, [\hat{\boldsymbol{t}}_j; \boldsymbol{c}_j]); \\ \boldsymbol{c}^j &= \text{attend}(\boldsymbol{c}_{1:n}, \hat{t}_{1:j}); \\ \hat{t}_j &\sim p(t_j | \hat{t}_{1:j-1}, \boldsymbol{x}_{1:n}). \end{aligned} \tag{17.7}$$

С точки зрения репрезентативной способности эта архитектура включает в себя рассмотренную выше архитектуру кодировщик–декодер: если положить $\text{attend}(\boldsymbol{c}_{1:n}, \hat{t}_{1:j}) = \boldsymbol{c}_n$, то получатся уравнения (17.6).

Как выглядит функция attend(·, ·)? Вы уже, наверное, догадались, что это обучаемая параметрическая функция. В этой книге мы следуем идее механизма внимания, изложенной в работе Bahdanau et al. [2014], где он впервые был применен в контексте генерации последовательностей из последовательностей[1]. Хотя этот конкретный механизм внимания завоевал популярность и работает хорошо, возможны многочисленные варианты. В работе Luong et al. [2015] некоторые из них исследуются в контексте машинного перевода.

Реализованный механизм внимания *мягкий*, т. е. на каждом этапе декодер видит взвешенное среднее векторов $\boldsymbol{c}_{1:n}$, в котором веса выбираются механизмом внимания.

---

[1] Наше описание декодера в некоторых мелких деталях отличается от приведенного в работе Bahdanau et al. [2014] и больше похоже на декодер из работы Luong et al. [2015].

Формально на этапе j мягкий механизм внимания порождает смесь векторов $\boldsymbol{c}^j$:

$$\boldsymbol{c}^j = \sum_{i=1}^{n} \boldsymbol{\alpha}^j_{[i]} \cdot \boldsymbol{c}_i.$$

Здесь $\boldsymbol{\alpha}^j \in \mathbb{R}^n_+$ – вектор весов внимания для j-го этапа, все элементы которого $\boldsymbol{\alpha}^j_{[i]}$ положительны и в сумме равны 1.

Значения $\boldsymbol{\alpha}^j_{[i]}$ порождаются двухэтапным процессом. Сначала сеть прямого распространения MLP^{att} порождает ненормированные веса внимания $\bar{\boldsymbol{\alpha}}^j_{[i]}$ с учетом состояния декодера в момент j и каждого из векторов $\boldsymbol{c}^i$:

$$\begin{aligned}\bar{\boldsymbol{\alpha}}^j &= \bar{\boldsymbol{\alpha}}^j_{[1]}, \ldots, \bar{\boldsymbol{\alpha}}^j_{[n]} = \\ &= \text{MLP}^{\text{att}}([\boldsymbol{s}_j; \boldsymbol{c}_1]), \ldots, \text{MLP}^{\text{att}}([\boldsymbol{s}_j; \boldsymbol{c}_n]).\end{aligned} \tag{17.8}$$

Затем ненормированные веса $\bar{\boldsymbol{\alpha}}^j_{[i]}$ нормируются с помощью функции softmax, чтобы получилось распределение вероятностей.

В контексте машинного перевода MLP^{att} можно интерпретировать как вычисление *мягкого выравнивания* между текущим состоянием декодера $\boldsymbol{s}_j$ (улавливающим недавно порожденные слова на иностранном языке) и всеми компонентами $\boldsymbol{c}_i$ исходного предложения.

Тогда полная функция внимания имеет вид:

$$\begin{aligned}\text{attend}(\boldsymbol{c}_{1:n}; \hat{t}_{1:j}) &= \boldsymbol{c}^j; \\ \boldsymbol{c}^j &= \sum_{i=1}^{n} \boldsymbol{\alpha}^j_{[i]} \cdot \boldsymbol{c}_i; \\ \boldsymbol{\alpha}^j &= \text{softmax}(\bar{\boldsymbol{\alpha}}^j_{[1]}, \ldots, \bar{\boldsymbol{\alpha}}^j_{[n]}); \\ \bar{\boldsymbol{\alpha}}^j_{[i]} &= \text{MLP}^{\text{att}}([\boldsymbol{s}_j; \boldsymbol{c}_i]),\end{aligned} \tag{17.9}$$

а весь процесс генерации последовательность-в-последовательность описывается уравнениями:

$$\begin{aligned}p(t_{j+1} = k | \hat{t}_{1:j}, \boldsymbol{x}_{1:n}) &= f(O_{\text{dec}}(\boldsymbol{s}_{j+1})); \\ \boldsymbol{s}_{j+1} &= R_{\text{dec}}(\boldsymbol{s}_j, [\hat{\boldsymbol{t}}_j;\ \boldsymbol{c}^j]); \\ \boldsymbol{c}^j &= \sum_{i=1}^{n} \boldsymbol{\alpha}^j_{[i]} \cdot \boldsymbol{c}_i; \\ \boldsymbol{c}_{1:n} &= \text{biRNN}^*_{\text{enc}}(\boldsymbol{x}_{1:n}); \\ \boldsymbol{\alpha}^j &= \text{softmax}(\bar{\boldsymbol{\alpha}}^j_{[1]}, \ldots, \bar{\boldsymbol{\alpha}}^j_{[n]}); \\ \bar{\boldsymbol{\alpha}}^j_{[i]} &= \text{MLP}^{\text{att}}([\boldsymbol{s}_j; \boldsymbol{c}_i]), \\ \hat{t}_j &\sim p(t_j | \hat{t}_{1:j-1}, \boldsymbol{x}_{1:n}); \\ f(\boldsymbol{z}) &= \text{softmax}(\text{MLP}^{\text{out}}(\boldsymbol{z})); \\ \\ \text{MLP}^{\text{att}}([\boldsymbol{s}_j; \boldsymbol{c}_i]) &= \boldsymbol{v}\tanh([\boldsymbol{s}_j; \boldsymbol{c}_i]\boldsymbol{U} + \boldsymbol{b}).\end{aligned} \tag{17.10}$$

Схематическое изображение этой архитектуры показано на рис. 17.5.

Зачем использовать biRNN-кодировщик для преобразования обусловливающего предложения $\boldsymbol{x}_{1:n}$ в контекстные векторы $\boldsymbol{c}_{1:n}$, а не дать механизму внима-

ния доступ непосредственно к $\boldsymbol{x}_{1:n}$? Почему бы не взять просто $\boldsymbol{c}^j = \sum_{i=1}^{n} \boldsymbol{\alpha}^j_{[i]}$ и $\bar{\boldsymbol{\alpha}}^j_{[i]} = \text{MLP}^{\text{att}}([\boldsymbol{s}_j, \boldsymbol{x}_i])$? Можно было бы поступить и так, но процесс кодирования дает нам важные преимущества. Во-первых, порождаемые biRNN векторы $\boldsymbol{c}_i$ представляют объекты $\boldsymbol{x}_i$ в *их последовательном контексте*, т. е. представляют окно, окружающее объект $\boldsymbol{x}_i$, а не сам этот объект. Во-вторых, если кодировщик обучается совместно с декодером, то оба они эволюционируют вместе, и сеть может обучиться кодировать релевантные свойства входа, полезные для декодирования, которые, возможно, не встречаются в исходной последовательности $\boldsymbol{x}_{1:n}$ непосредственно. Например, biRNN-кодировщик может обучиться кодировать *позицию* $\boldsymbol{x}_i$ внутри последовательности, а декодер мог бы использовать эту информацию для доступа к элементам по порядку или уделять элементам в начале последовательности больше внимания, чем элементам в конце.

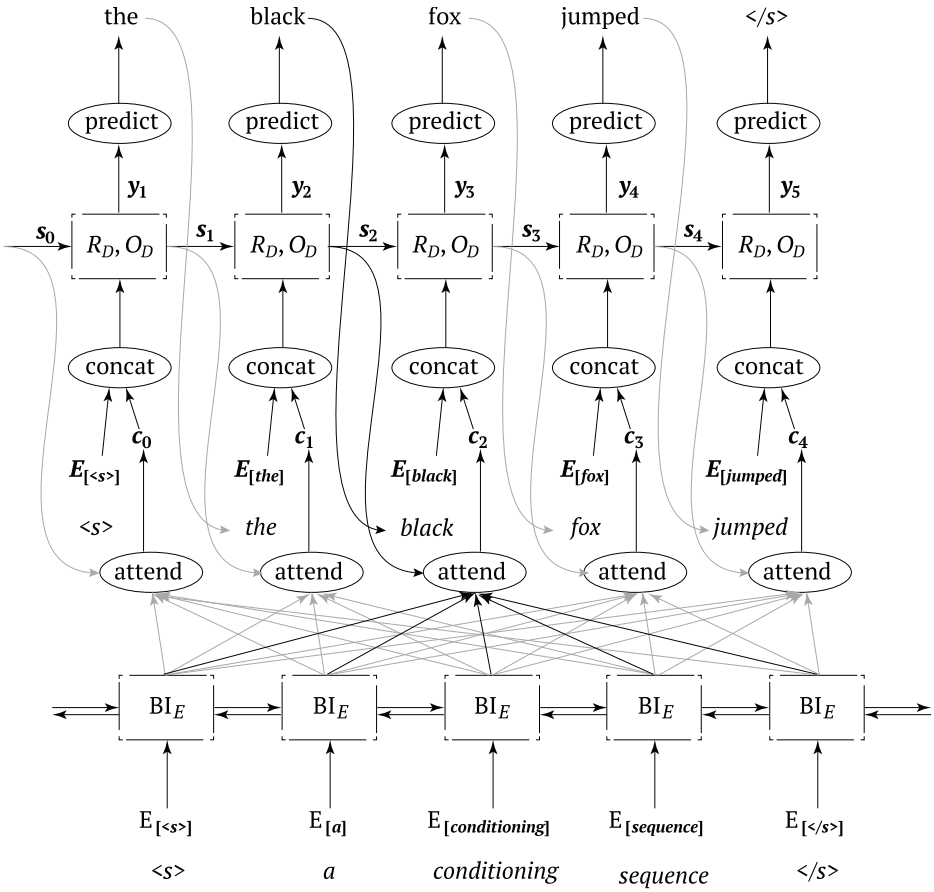


Рис. 17.5 ❖ RNC-генератор последовательность-в-последовательность с механизмом внимания

Модели условной генерации с вниманием (*аттентивные* модели) очень эффективны и хорошо работают для многих задач генерации последовательности из последовательности.

17.4.1. Вычислительная сложность

Условная генерация *без* внимания обходится сравнительно дешево: время кодирования линейно зависит от длины входа ($O(n)$), а время декодирования линейно зависит от длины выхода ($O(m)$). Правда, генерация распределения слов из большого словаря сама по себе стоит дорого, но это никак не связано с нашим анализом, в котором обработка словаря считается операцией, занимающей постоянное время. Таким образом, полная сложность генерации последовательности из последовательности составляет $O(m + n)$[1].

А какова стоимость добавления механизма внимания? Кодирование входной последовательности по-прежнему требует времени $O(n)$. Но на каждом шаге процесса декодирования теперь необходимо вычислять $\boldsymbol{c}^i$. Это влечет за собой n вычислений MLP^{att}, за которыми следуют шаг нормировки и суммирование n векторов. Сложность шага декодирования уже не постоянна, а линейно зависит от длины обусловливающей последовательности ($O(n)$), так что полное время работы составляет $O(m \times n)$.

17.4.2. Возможность интерпретации

Сети кодировщик–декодер без механизма внимания (как и большинство других нейросетевых архитектур) в высшей степени непрозрачны: мы не понимаем, что именно закодировано в векторе, как эта информация используется декодером и что является причиной конкретного поведения декодера. Важное достоинство аттентивной архитектуры состоит в том, что она дает простой способ заглянуть внутрь некоторых рассуждений декодера и узнать, чему он обучился. На каждом этапе процесса декодирования можно посмотреть на вычисленные веса внимания α^i и увидеть, какие участки исходного предложения декодер счел релевантными при порождении выхода. Это, конечно, слабая форма интерпретируемости, но и она куда лучше полной непрозрачности моделей без внимания.

17.5. Модели на основе внимания в NLP

Условная генерация с вниманием – очень мощная архитектура. Это основной алгоритм в современных системах машинного перевода, и он же дает сильные результаты во многих других задачах NLP. В этом разделе мы приведем несколько примеров его использования.

17.5.1. Машинный перевод

Первоначально мы описывали машинный перевод в контексте простой генерации последовательности из последовательности, но в современных системах используются модели с механизмом внимания.

Первые результаты применения моделей типа последовательность-в-последовательность с механизмом внимания были описаны в работе Bahdanau et al.

[1] Длина выхода m, в принципе, не ограничена, но на практике декодеры обучаются порождать выход, для которого распределение длин примерно такое же, как в обучающем наборе данных, а в худшем случае всегда можно задать жесткое ограничение на длину сгенерированной последовательности.

[2014], где использовалась, по существу, та же архитектура, которую мы описали в предыдущем разделе (с РНС типа GRU), а при генерации декодером на этапе тестирования применялся лучевой поиск. Хотя в работе Luong et al. [2015] изучались варианты механизма внимания, дающие некоторый выигрыш, наибольшего прогресса в нейросетевом переводе удалось добиться благодаря исходной архитектуре последовательность-в-последовательность с вниманием (на основе РНС типа LSTM или GRU), но с измененными входами.

Конечно, нельзя ожидать, что в этом коротком разделе мы сумеем охватить всю тематику нейросетевого машинного перевода, но все же перечислим некоторые усовершенствования, достигнутые Сеннрихом с коллегами, которым удалось раздвинуть границы возможного и установить новый отраслевой стандарт.

СУБСЛОВЕСНЫЕ ЕДИНИЦЫ Для работы со словарями языков с развитым словоизменением (а также для того чтобы ограничить размер словаря в общем случае) в работе Sennrich et al. [2016a] предложено перейти к субсловесным единицам, меньшим одной лексемы. Текст на исходном и целевом языках обрабатывается алгоритмом Bpe, который ищет наиболее значимые субсловесные единицы (сам алгоритм описан в конце раздела 10.5.5). Для английского языка этот этап, скорее всего, найдет такие единицы, как er, est, un, low и wid. Затем слова в предложениях на обоих языках разбиваются в соответствии с индуцированной сегментацией (например, словосочетание widest network преобразуется в wid_ _est net_ _work). Обработанный таким образом корпус подается на вход алгоритма обучения сети последовательность-в-последовательность с вниманием. После декодирования тестовых предложений выход обрабатывается еще раз, чтобы образовать из субсловесных единиц слова. Этот процесс уменьшает количество неизвестных лексем, облегчает обобщаемость на новые элементы словаря и повышает качество перевода. В том же русле ведутся исследования по работе непосредственно на уровне литер (когда кодируются и декодируются не слова, а литеры); значительного успеха в этом направлении удалось добиться в работе [Chung et al., 2016].

ПОКЛЮЧЕНИЕ ОДНОЯЗЫЧНЫХ ДАННЫХ Модели типа последовательность-в-последовательность обучаются на параллельных корпусах выровненных предложений на исходном и целевом языках. Такие корпусы существуют, но, естественно, их меньше, чем одноязычных данных, объем которых практически бесконечен. На самом деле в предыдущем поколении систем статистического машинного перевода[1] *модель перевода* обучалась на параллельных данных, а отдельная языковая модель – на гораздо более обширных одноязычных данных. Архитектура последовательность-в-последовательность пока не допускает такого разделения, а обучение языковой модели (декодер) и модели перевода (взаимодействие кодировщика и декодера) производится совместно.

Как можно воспользоваться данными на языке перевода в системе последовательность-в-последовательность? В работе Sennrich et al. [2016b] предлагается следующий протокол перевода: при попытке перевода с исходного на целевой язык сначала обучить модель перевода с целевого языка на исходный и использовать ее для перевода большого одноязычного корпуса предложений на целевом

[1] Обзор *см.* в книге Koehn [2010], а также в книге, посвященной машинному переводу по синтаксису, вышедшей в этой серии [Williams et al., 2016].

языке. Затем добавить получившиеся пары (целевой, исходный) в параллельный корпус в виде примеров (исходный, целевой). Обучить систему машинного перевода с исходного языка на целевой на таком комбинированном корпусе. Заметим, что хотя теперь система обучается на автоматически созданных примерах, все предложения на целевом языке, которые она видит, оригинальны, так что компонент языкового моделирования никогда не обучается на автоматически порожденных текстах. Хотя этот протокол выглядит не вполне честной уловкой, он существенно повышает качество перевода. В ходе дальнейших исследований, вероятно, появятся более чистые способы поключения одноязычных данных.

ЛИНГВИСТИЧЕСКИЕ АННОТАЦИИ Наконец, в работе Sennrich and Haddow [2016] показано, что архитектура последовательность-в-последовательность с вниманием может обучить улучшенную модель перевода, если входные данные дополнить лингвистическими аннотациями. Это означает, что, получив исходное предложение $w_1, \ldots, w_n$, мы создаем входные векторы $\boldsymbol{x}_{1:n}$, не просто сопоставляя вектор погружения каждому слову ($\boldsymbol{x}_i = \boldsymbol{E}_{[w_i]}$), а пропускаем предложение через конвейер лингвистического аннотирования, который включает частеречную разметку, разбор синтаксических зависимостей и лемматизацию. Затем каждое слово дополняется кодирующим вектором, содержащим его метку части речи ($\boldsymbol{p}_i$), его метку зависимости от заглавного слова ($\boldsymbol{r}_i$), его лемму ($\boldsymbol{l}_i$) и морфологические признаки ($\boldsymbol{m}_i$). Тогда входные векторы $\boldsymbol{x}_{1:n}$ определяются как конкатенация всех этих признаков: $\boldsymbol{x}_i = [\boldsymbol{w}_i; \boldsymbol{p}_i; \boldsymbol{r}_i; \boldsymbol{l}_i; \boldsymbol{m}_i]$. Эти дополнительные признаки всякий раз повышают качество перевода, что доказывает полезность лингвистической информации даже при наличии мощной модели, которая теоретически может обучиться лингвистическим концепциям самостоятельно. Аналогично в работе Aharoni and Goldberg [2017] показано, что, обучив декодер системы перевода с немецкого на английский порождать линеаризованные синтаксические деревья вместо последовательности слов, можно добиться более единообразного изменения порядка слов и улучшить качество перевода. Эти работы лишь едва затрагивают проблематику интеграции лингвистической информации. В будущем, возможно, появятся дополнительные лингвистические признаки, допускающие интеграцию, или будут найдены новые способы интеграции.

ОТКРЫТЫЕ ВОПРОСЫ На момент написания этой книги основными открытыми вопросами нейросетевого машинного перевода были: масштабирование размера выходного словаря (или исключение зависимости от него путем перехода к литерному выходу), обучение с учетом лучевого поиска при декодировании и ускорение обучения и декодирования. Еще одна набирающая популярность тема – переход к моделям, в которых используется синтаксическая информация. Но и вообще эта область исследования быстро развивается, так что к моменту выхода книги из печати этот абзац, возможно, утратит актуальность.

17.5.2. Морфологическое словоизменение

Задача морфологического словоизменения, которая выше обсуждалась в контексте моделей последовательность-в-последовательность, также работает лучше, если включить в архитектуру механизм внимания, что было наглядно продемонстрировано на примере системы, победившей в конкурсе SIGMORPHON по морфологической рефлексии [Cotterell et al., 2016]. В победившей системе

[Kann and Schütze, 2016] использовалась готовая модель последовательность-в-последовательность с механизмом внимания. На вход подавалось слово и желаемое словоизменение, заданное в виде списка меток частей речи и морфологических признаков, например: NOUN Gender=Male Number=Plural, а на выходе следовало получить измененную форму. Чтобы сформулировать задачу как модель последовательность-в-последовательность, создается входная последовательность, содержащая список сведений о словоизменении, и список литер входного слова. На выходе получается список литер целевого слова.

17.5.3. Синтаксический анализ

Хотя для этой задачи существуют более подходящие архитектуры, в работе Vinyals et al. [2014] показано, что модели последовательность-в-последовательность с вниманием могут давать конкурентоспособные результаты синтаксического анализа посредством чтения предложения (слово за словом) и вывода последовательности решений о расстановке скобок. Может показаться, что эта архитектура не идеальна для анализа – и действительно, специализированные архитектуры позволяют получить лучшие результаты, что доказывает работа Cross and Huang [2016a]. Но, принимая во внимание общность архитектуры, система работает на удивление хорошо и порождает впечатляющие результаты анализа. Чтобы повысить конкурентоспособность, необходимо предпринять какие-то дополнительные действия, поскольку в таком виде архитектуре нужно очень много обучающих данных. Она обучается на деревьях разбора, порожденных двумя анализаторами, обученными на большом корпусе, из которых отобраны деревья, где оба анализатора дают одинаковые результаты (разборы с высоким уровнем достоверности). Кроме того, в окончательном анализаторе используется ансамбль (раздел 5.2.3) из нескольких сетей внимания.

Часть IV

ДОПОЛНИТЕЛЬНЫЕ ТЕМЫ

Глава 18

Моделирование деревьев с помощью рекурсивных нейронных сетей

РНС очень полезны для моделирования последовательностей. Но в лингвистической обработке часто возникает естественное желание работать с древовидными структурами. Это могут быть синтаксические деревья, деревья повествования и даже деревья, представляющие эмоциональную окраску различных частей предложения [Socher et al., 2013b]. Иногда нужно предсказывать значения на основе конкретных узлов дерева, на основе корневого узла или назначить оценку качества всему дереву или его части. Бывает и так, что сама структура дерева нас не интересует, а важны выводы об участках предложения. В таких случаях дерево служит просто вспомогательной структурой, помогающей управлять процессом кодирования последовательности в вектор фиксированного размера.

Абстракция *рекурсивной нейронной сети* (RecNN) [Pollack, 1990], популяризированная в NLP Ричардом Сочером с коллегами [Socher, 2014, Socher et al., 2010, 2011, 2013a], – это обобщение РНС с последовательностью на (двоичные) деревья[1].

Как РНС кодирует каждый префикс предложения в виде вектора состояния, так и RecNN кодирует каждый узел дерева в виде вектора состояния из $\mathbb{R}^d$. Затем эти векторы состояния можно использовать либо для предсказания значений соответствующих узлов, либо для назначения качества каждому узлу, либо как семантическое представление участков предложения с корнем в узлах.

Основное интуитивное соображение, лежащее в основе рекурсивных нейронных сетей, заключается в том, что каждое поддерево представляется d-мерным вектором, а представление узла p с дочерними узлами c_1 и c_2 является функцией от представления этих узлов: $vec(p) = f(vec(c_1), vec(c_2))$, где f – функция композиции, принимающая два d-мерных вектора и возвращающая один d-мерный вектор. Если в РНС состояние $\boldsymbol{s}_i$ служит для кодирования всей последовательности $\boldsymbol{x}_{1:i}$, то в RecNN состояние, ассоциированное с вершиной p, кодирует все поддерево с корнем в этой вершине (*см*. рис. 18.1).

[1] Хотя эти концепции представлены в терминах двоичных деревьев разбора, они легко переносятся на рекурсивно определенные структуры общего вида, а основная техническая трудность – определить эффективную форму комбинационной функции $\boldsymbol{R}$.

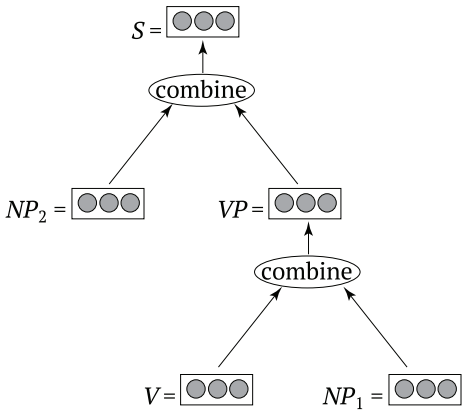
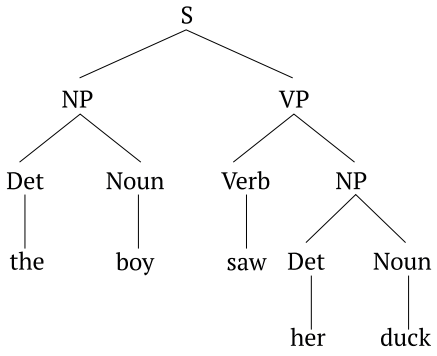


Рис. 18.1 ❖ Рекурсивная нейронная сеть.
Представления V и NP1 объединяются в представление VP.
Затем представления VP и NP2 объединяются в представление S

18.1. Формальное определение

Рассмотрим двоичное дерево разбора T предложения из n слов. Напомним, что упорядоченное непомеченное дерево над строкой $x_1, \ldots, x_n$ можно представить в виде однозначно определенного множества троек (i, k, j) – таких, что $i \le k \le j$. Каждая такая тройка означает, что узел, охватывающий участок слов $x_{i:j}$, является родителем узлов, охватывающих участки $x_{i:k}$ и $x_{k+1:j}$. Тройки вида (i, i, i) соответствуют терминальным символам в листьях дерева (словам x_i). Переходя от непомеченного случая к помеченному, мы можем представить дерево в виде множества шестикортежей $(A \to B, C, i, k, j)$, где i, k и j, как и раньше, обозначают линейные участки, а A, B и C – метки узлов, охватывающих участки $x_{i:j}$, $x_{i:k}$ и $x_{k+1:j}$ соответственно. Здесь листовые узлы имеют вид $(A \to A, A, i, i, i)$, где A – предтерминальный символ. Такие кортежи называются *продукциями*. Рассмотрим, к примеру, синтаксическое дерево предложения «the boy saw her duck» (мальчик увидел ее утку).



Соответствующие ему непомеченное и помеченное представления показаны в табл. 18.1.

Таблица 18.1. Непомеченное и помеченное представления

Непомеченное	Помеченное	Соответствующий участок
(1,1,1)	(Det, Det, Det, 1, 1, 1)	$x_{1:1}$ the
(2.2.2)	(Nound, Noun, Noun, 2, 2, 2)	$x_{2:2}$ boy
(3,3,3)	(Verb, Verb, Verb, 3, 3, 3)	$x_{3:3}$ saw
(4, 4, 4)	(Det, Det, Det, 4, 4, 4)	$x_{4:4}$ her
(5, 5, 5)	(Noun, Noun, Noun, 5, 5, 5)	$x_{5:5}$ duck
(4, 4, 5)	(NP, Det, Noun, 4, 4, 5)	$x_{4:5}$ her duck
(3, 3, 5)	(VP, Verb, NP, 3, 3, 5)	$x_{3:5}$ saw her duck
(1, 1, 2)	(NP, Det, Nound, 1, 1, 2)	$x_{1:2}$ the boy
(1, 2, 5)	(S, NP, VP, 1 2, 5)	$x_{1:5}$ the boy saw her duck

Показанное выше множество продукций можно единственным способом преобразовать во множество узлов дерева $q^A_{i:j}$ (так обозначается узел с символом A, охватывающий участок $x_{i:j}$), просто игнорируя элементы (B, C, k) в каждой продукции. Теперь мы готовы определить рекурсивную нейронную сеть.

Рекурсивная нейронная сеть (RecNN) – это функция, которая принимает дерево разбора предложения из n слов $x_1, \ldots, x_n$. Каждое слово представляется d-мерным вектором $\boldsymbol{x_i}$, а дерево – множеством $\mathcal{T}$ продукций ($A \to B, C, i, j, k$). Введем для узлов $\mathcal{T}$ обозначение $q^A_{i:j}$. RecNN возвращает на выходе множество *векторов внутреннего состояния* $\boldsymbol{s^A_{i:j}} \in \mathbb{R}^d$, каждый из которых представляет соответствующий узел дерева $q^A_{i:j}$ и кодирует все поддерево с корнем в этом узле. Как и линейная РНС, древовидная RecNN определяется рекурсивно с помощью функции $\boldsymbol{R}$, которая определяет внутренний вектор данного узла через внутренние векторы его непосредственных потомков[1]. Формально:

$$\begin{aligned} \text{RecNN}(x_1, \ldots, x_n, \mathcal{T}) &= \{\boldsymbol{s^A_{i:j}} \in \mathbb{R}^d \mid q^A_{i:j} \in \mathcal{T}\}; \\ \boldsymbol{s^A_{i:i}} &= v(x_i); \\ \boldsymbol{s^A_{i:j}} &= R(A, B, C, \boldsymbol{s^B_{i:k}}, \boldsymbol{s^C_{k+1:j}}), \quad q^B_{i:k} \in \mathcal{T}, q^C_{k+1:j} \in \mathcal{T}. \end{aligned} \tag{18.1}$$

Функция $\boldsymbol{R}$ обычно принимает вид простого линейного преобразования, за которым может следовать нелинейная функция активации g:

$$R(A, B, C, \boldsymbol{s^B_{i:k}}, \boldsymbol{s^C_{k+1:j}}) = g([\boldsymbol{s^B_{i:k}}; \boldsymbol{s^C_{k+1:j}}]\boldsymbol{W}). \tag{18.2}$$

В этой формулировке $\boldsymbol{R}$ игнорирует метки дерева, т. к. используется одна и та же матрица $\boldsymbol{W} \in \mathbb{R}^{2d \times d}$ для всех комбинаций. Это может быть полезно в случае, когда меток узлов не существует (например, когда дерево не представляет синтаксическую структуру с четко определенными метками) или когда они ненадежны. Однако если метки доступны, то, вообще говоря, полезно учитывать их в функ-

[1] В работе Le and Zuidema [2014] определение RecNN обобщается, так что у каждого узла, помимо вектора внутреннего состояния, есть еще *вектор внешнего состояния*, представляющий всю структуру вокруг поддерева с корнем в этом узле. Эта формулировка основана на рекурсивном вычислении с помощью классического алгоритма максимизации правдоподобия (inside-outside algorithm) и может рассматриваться как аналог biRNN для древовидной RecNN.

ции композиции. Одно из возможных решений – ввести *погружение меток* $v(A)$, отображающее каждый нетерминальный символ в d_{nt}-мерный вектор, и изменить $\boldsymbol{R}$, включив погружения символов в функцию композиции:

$$R(A, B, C, \boldsymbol{s}^{B}_{i:k}, \boldsymbol{s}^{C}_{k+1:j}) = g([\boldsymbol{s}^{B}_{i:k}; \boldsymbol{s}^{C}_{k+1:j}; v(A); v(B)]\boldsymbol{W}) \tag{18.3}$$

(здесь $\boldsymbol{W} \in \mathbb{R}^{2d+2d_{nt}\times d}$). Такой подход принят в работе Qian et al. [2015]. Авторы работы Socher et al. [2013a] поступили по-другому, решив использовать разные матрицы композиции для каждой пары символов B, C:[1]

$$R(A, B, C, \boldsymbol{s}^{B}_{i:k}, \boldsymbol{s}^{C}_{k+1:j}) = g([\boldsymbol{s}^{B}_{i:k}; \boldsymbol{s}^{C}_{k+1:j}]\boldsymbol{W}^{BC}). \tag{18.4}$$

Такая формулировка полезна, когда количество нетерминальных символов (или количество возможных комбинаций символов) относительно мало, как обычно бывает в деревьях разбора грамматики непосредственных составляющих. Похожая модель использовалась в работе Hashimoto et al. [2013] для кодирования поддеревьев в задаче классификации семантических связей.

18.2. Обобщения и вариации

Поскольку все приведенные выше определения $\boldsymbol{R}$ подвержены проблеме исчезающих градиентов, характерной для простой РНС, несколько авторов исследовало возможность заменить ее функциями в духе вентильной архитектуры LSTM, что привело к древовидным LSTM-сетям [Tai et al., 2015, Zhu et al., 2015b]. Вопрос об оптимальном представлении дерева по-прежнему остается открытой проблемой, и обширное пространство возможных комбинационных функций $\boldsymbol{R}$ еще только предстоит исследовать. Предлагались также другие варианты древовидных РНС: *рекурсивная матрично-векторная модель* [Socher et al., 2012] и *рекурсивная нейронная тензорная сеть* [Socher et al., 2013b]. В первом варианте каждое слово представляется в виде комбинации вектора и матрицы, где вектор, как и раньше, определяет статическое семантическое содержание слова, а матрица выступает в роли обученного «оператора» для слова, что открывает возможность для более тонких семантических композиций, чем сложение и взвешенное усреднение, вытекающее из конкатенации, сопровождаемой линейным преобразованием. Во втором варианте слова ассоциируются с векторами, как обычно, но функция композиции становится более выразительной, поскольку основывается не на матричных, а на тензорных операциях.

В своей собственной работе [Kiperwasser and Goldberg, 2016a] мы предлагаем кодировщик дерева, который может работать с произвольными, а не только двоичными деревьями. Кодирование основано на РНС (точнее, на LSTM-сетях), когда кодировка каждого поддерева рекурсивно определяется как объединение состояний двух РНС, одна из которых кодирует левые поддеревья (слева направо) и заканчивается в корневом узле, а другая – правые поддеревья (справа налево) и заканчивается в корневом узле.

[1] Тривиальное обобщение этого подхода, не исследованное в литературе, – обусловить матрицу преобразования также символом $\boldsymbol{A}$.

18.3. Обучение рекурсивных нейронных сетей

Процедура обучения рекурсивной нейронной сети устроена по тому же рецепту, что и обучение сетей других типов: определить функцию потерь, составить граф вычислений, вычислить градиенты, применяя обратное распространение[1], и обучить параметры с помощью СГС.

Что касается функции потерь, то, как и в линейной РНС, мы можем ассоциировать потерю либо с корнем дерева, либо с любым заданным узлом, либо с множеством узлов, причем в последнем случае потери в отдельных узлах объединяются, обычно посредством суммирования. Функция потерь основана на помеченных обучающих данных, в которых метки или еще какие-то величины ассоциированы с различными узлами дерева.

Кроме того, RecNN можно рассматривать как кодировщик, а вектором внутреннего состояния, ассоциированным с некоторым узлом, считается кодировка дерева с корнем в этом узле. Кодирование потенциально может быть чувствительно к произвольным свойствам структуры. Затем вектор передается на вход другой сети.

Дополнительные сведения о рекурсивных нейронных сетях и их использовании в задачах обработки естественного языка *см*. в докторской диссертации Socher [2014].

18.4. Простая альтернатива – линеаризованные деревья

Абстракция RecNN предлагает гибкий механизм кодирования деревьев в виде векторов с помощью рекурсивного композиционного подхода. RecNN кодирует не только данное дерево, но и все его поддеревья. Если такая рекурсивность кодирования не нужна и нам всего лишь требуется получить векторное представление всего дерева, которое было бы чувствительно к его структуре, то существуют и более простые альтернативы. В частности, в нескольких работах показана высокая эффективность линеаризации деревьев, т. е. преобразования их в последовательность, которая подается на вход вентильной РНС-приемщику (или biRNN-кодировщику) [Choe and Charniak, 2016, Luong et al., 2016, Vinyals et al., 2014]. Конкретно, дерево предложения *the boy saw her duck*, приведенного выше в качестве примера, будет преобразовано в линейную строку:

(S (NP (Det the Det) (Noun boy Noun) NP) (VP (Verb saw Verb) (NP (Det her Det) (Noun duck Noun) NP) VP) S),

которая затем будет подана на вход вентильной РНС, например LSTM. Тогда конечное состояние РНС можно использовать как векторное представление дерева. Или же древовидную структуру можно оценить, обучив РНС языковой модели на таких линеаризованных деревьях разбора и приняв вычисленную языковой моделью вероятность линеаризованного дерева за оценку его качества.

---

[1] До появления абстракции графа вычислений процедура обратного распространения для вычисления градиентов конкретно в RecNN называлась алгоритмом обратного распространения через структуру (BPTS) [Goller and Küchler, 1996].

18.5. Перспективы

Мощная и интригующая концепция рекурсивных древовидных сетей кажется идеально соответствующей рекурсивной природе языка. Однако по состоянию на конец 2016 года можно смело сказать, что такие сети еще не продемонстрировали явного и неоспоримого преимущества по сравнению более простыми архитектурами. На самом деле во многих случаях линейные модели типа PHC улавливают требуемые закономерности ничуть не хуже. Либо мы еще не встретили приложения, которое раскрыло бы все возможности древовидных сетей, либо пока не нашли правильной архитектуры или режимов обучения. Сравнение и анализ применения древовидных и линейных сетей в лингвистических задачах можно найти в работе Li et al. [2015]. На данный момент использование древовидных сетей для обработки языковых данных все еще является темой активных исследований. Найти подходящее приложение для таких сетей, предложить улучшенный режим обучения или показать, что древовидные архитектуры не нужны вовсе, – все это увлекательные направления научной работы.

Глава 19

Предсказание структурного выхода

Во многих проблемах NLP выходные данные имеют некоторую структуру, т. е. являются не просто меткой класса или распределением меток класса, а последовательностью, деревом или графом. Канонические примеры – разметка последовательности (в том числе частеречная разметка), сегментация последовательности (разбиение на блоки, распознавание именованных сущностей), синтаксический анализ и машинный перевод. В этой главе мы обсудим применение нейросетевых моделей к таким структурным задачам.

19.1. Структурное предсказание на основе поиска

Типичный подход к предсказанию структурированных данных основан на поиске. Подробное обсуждение предсказания структуры на основе поиска в системах NLP, предшествующих глубокому обучению, *см*. в книге Smith [2011]. Описанные в ней методы легко адаптируются к нейронным сетям. В литературе по нейронным сетям такие модели обсуждаются в контексте *обучения на основе энергии* [LeCun et al., 2006, раздел 7]. Здесь для их представления использованы конфигурация и терминология, принятые в сообществе NLP. Структурное предсказание на основе поиска формулируется как задача поиска по всем возможным структурам:

$$\text{predict}(x) = \underset{y \in \mathcal{Y}(x)}{\text{argmax}} \, \text{score}_{\text{global}}(x, y), \tag{19.1}$$

где x – входная структура, y – выход, зависящий от x (в типичном примере x – предложение, а y – его частеречная разметка или дерево разбора), $\mathcal{Y}(x)$ – множество всех допустимых структур для данного x, и мы ищем такой выход y, который доставляет максимум оценке пары x, y.

19.1.1. Структурное предсказание с помощью линейных моделей

В обширной литературе по структурному предсказанию с помощью линейных и логлинейных моделей оценивание моделируется линейной функцией:

$$\text{score}_{\text{global}}(x, y) = \boldsymbol{w} \cdot \Phi(x, y), \tag{19.2}$$

где Φ – функция выделения признаков, а $\boldsymbol{w}$ – вектор весов.

Чтобы поиск оптимума y не оказался вычислительно неразрешимой задачей, структура y разлагается на части, а функция выделения признаков определяется в терминах частей, где $\phi(p)$ – локальная функция выделения признаков:

$$\Phi(x, y) = \sum_{p \in \text{parts}(x,y)} \phi(p). \tag{19.3}$$

Каждая часть оценивается независимо, а полная оценка структуры равна сумме оценок компонент:

$$\text{score}_{\text{global}}(x, y) = \boldsymbol{w} \cdot \Phi(x, y) = \boldsymbol{w} \cdot \sum_{p \in y} \phi(p) = \boldsymbol{w} \cdot \sum_{p \in y} \phi(p) = \sum_{p \in y} \text{score}_{\text{local}}(p), \tag{19.4}$$

где $p \in y$ – краткая запись $p \in \text{parts}(x, y)$. Разложение y на части производится так, что существует алгоритм вывода, допускающий эффективный поиск структуры с наилучшей оценкой при известных оценках частей.

19.1.2. Нелинейное структурное предсказание

Безо всяких проблем линейную функцию оценки частей можно заменить нейронной сетью:

$$\text{score}_{\text{global}}(x, y) = \sum_{p \in y} \text{score}_{\text{local}}(p) = \sum_{p \in y} \text{NN}(\phi(p)), \tag{19.5}$$

где $\phi(p)$ отображает часть p в d_{in}-мерный вектор.

В случае сети прямого распространения с одним скрытым слоем имеем:

$$\text{score}_{\text{global}}(x, y) = \sum_{p \in y} \text{MLP}_1(\phi(p)) = \sum_{p \in y} (g(\phi(p)\boldsymbol{W}^1 + \boldsymbol{b}^1))\boldsymbol{w}, \tag{19.6}$$

где $\phi(p) \in \mathbb{R}^{d_{in}}$, $\boldsymbol{W}^1 \in \mathbb{R}^{d_{in} \times d_1}$, $\boldsymbol{b}^1 \in \mathbb{R}^{d_1}$, $\boldsymbol{w} \in \mathbb{R}^{d_1}$. Типичная цель структурного предсказания – сделать оценку золотой структуры y больше, чем любой другой структуры y', что приводит к такой функции потерь (обобщенный перцептрон [Collins, 2002]):

$$\max_{y'} \text{score}_{\text{global}}(x, y') - \text{score}_{\text{global}}(x, y). \tag{19.7}$$

Для максимизации применяется специальный алгоритм поиска, в основе которого часто лежит динамическое программирование или похожая техника.

С точки зрения реализации, это означает следующее: создать граф вычислений CG_p для каждой возможной части и вычислить ее оценку. Затем выполнить логический вывод (т. е. поиск) для нахождения структуры y' с наилучшей оценкой, исходя из оценок ее частей. Соединить выходные узлы графов вычислений, соответствующие частям золотой (предсказанной) структуры y (y'), с узлом суммирования CG_y (CG'_y). Соединить CG_y и CG'_y, используя узел «минус» CG_l, и вычислить градиенты.

В работе LeCun et al. [2006, раздел 5] высказано соображение, что обобщенный перцептрон, возможно, не является хорошей функцией потерь для обучения нейронных сетей структурного предсказания, поскольку не оставляет зазора, и лучше бы использовать кусочно-линейную потерю с зазором:

$$\max(0, m + \max_{y' \neq y} \text{score}_{\text{global}}(x, y') - \text{score}_{\text{global}}(x, y)). \tag{19.8}$$

Описанная выше реализация тривиально модифицируется для работы с кусочно-линейной потерей.

Заметим, что в обоих случаях мы теряем полезные свойства линейной модели. В частности, модель перестает быть выпуклой. Этого следовало ожидать, потому что даже простейшая нелинейная нейронная сеть уже невыпукла. Тем не менее мы все же могли бы использовать стандартные методы оптимизации нейронных сетей для обучения структурной модели.

Обучение и вывод производятся медленнее, поскольку необходимо обсчитывать нейронную сеть (и вычислять градиенты) по одному разу для каждой части, т. е. всего $|\text{parts}(x, y)|$ раз.

ОБУЧЕНИЕ, ДОПОЛНЕННОЕ ФУНКЦИЕЙ СТОИМОСТИ Структурное предсказание – обширная область, и мы в этой книге даже не будем пытаться охватить ее во всей полноте. Функции потерь, регуляризаторы и методы, описанные, например, в книге Smith [2011], по большей части легко переносятся на нейронные сети, правда, с потерей выпуклости и сопутствующих теоретических гарантий. Один метод, достойный отдельного упоминания, – *обучение, дополненное функцией стоимости* (cost augmented training), или, как его еще называют, *вывод, дополненный функцией потерь* (loss augmented inference). Его применение в линейных структурных предсказаниях дает скромный выигрыш, но, как обнаружила моя группа, он играет существенную роль в успешном обучении нейросетевых моделей структурного предсказания с помощью обобщенного перцептрона или функции потерь с зазором, особенно когда используются такие сильные экстракторы признаков, как РНС.

Назначение члена максимизации в уравнениях (19.7) и (19.8) состоит в том, чтобы найти структуру y', которая имеет высокую оценку в текущей модели и при этом неверна. Тогда потеря отражает разность между оценками y' и золотой структуры y. После того как модель будет достаточно хорошо обучена, неверная структура y' и верная структура y, вероятно, окажутся похожи (поскольку модель обучалась назначать высокие оценки разумно хорошим структурам). Напомним, что глобальная функция оценки на самом деле является суммой оценок локальных частей. Части, встречающиеся в обоих членах оценки (y' и y), взаимно уничтожаются и дают нулевой градиент для ассоциированных параметров сети. Если y и y' похожи друг на друга, то большинство частей будет встречаться в обоих местах и, стало быть, сократится, что приведет к очень маленькому обновлению на примере.

Идея обучения, дополненного функцией стоимости, заключается в том, чтобы изменить максимизацию таким образом, дабы производился поиск структур y', которые имеют хорошую оценку в модели и *к тому же относительно неверны* в том смысле, что имеют много неверных частей. Формально кусочно-линейная целевая функция заменяется на такую:

$$\max\left(0, m + \max_{y' \neq y} \left(\text{score}_{\text{global}}(x, y') + \rho\Delta(y, y')\right) - \text{score}_{\text{global}}(x, y)\right), \tag{19.9}$$

где ρ – скалярный гиперпараметр, показывающий важность Δ относительно модельной оценки, а $\Delta(y, y')$ – функция, подсчитывающая количество неверных частей в y', отсутствующих в y:

$$\Delta(y, y') = |\{p : p \in y', p \notin y\}|. \tag{19.10}$$

На практике новый способ максимизации можно реализовать, увеличив локальную оценку каждой неверной части на ρ перед вызовом процедуры максимизации.

Обучение, дополненное функцией стоимости, выявляет совсем неверные примеры и оставляет в функции потерь больше несокращающихся членов, повышая тем самым эффективность обновления градиентов.

19.1.3. Вероятностная целевая функция (CRF)

Описанные выше функции потерь, основанные на ошибке и на зазоре, пытаются давать верной структуре оценку выше, чем неверной, но ничего не говорят об упорядочении структур с оценкой ниже максимальной или о разностях оценок между ними.

С другой стороны, дискриминантная вероятностная функция потерь пытается назначить вероятность каждой возможной структуре при условии известного входа, так чтобы вероятность правильной структуры оказалась максимальной. Вероятностная потеря учитывает оценки всех возможных структур, а не только наивысшую.

При вероятностном подходе, который известен также под названием *условные случайные поля* (conditional random fields – CRF), оценки всех частей рассматриваются как *потенциал клики* (*см*. Lafferty et al. [2001], Smith [2011]), а оценка каждой структуры y определяется по формуле:

$$\begin{aligned} \text{score}_{\text{CRF}}(x, y) = P(y \mid x) &= \frac{e^{\text{score}_{\text{global}}(x,y)}}{\sum_{y' \in \mathcal{Y}(x)} e^{\text{score}_{\text{global}}(x,y')}} \\ &= \frac{\exp \sum_{p \in y} \text{score}_{\text{local}}(p)}{\sum_{y' \in \mathcal{Y}(x)} \exp\left(\sum_{p \in y'} \text{score}_{\text{local}}(p)\right)} \\ &= \frac{\exp\left(\sum_{p \in y} \text{NN}(\phi(p))\right)}{\sum_{y' \in \mathcal{Y}(x)} \exp\left(\sum_{p \in y'} \text{NN}(\phi(p))\right)}. \end{aligned} \tag{19.11}$$

Функция оценки определяет условное распределение $P(y|x)$, и мы хотим установить параметры сети так, чтобы условное логарифмическое правдоподобие корпуса $\sum_{(x_i, y_i) \in \text{training}} \log P(y_i|x_i)$ было максимально.

Потеря на заданном обучающем примере (x, y) тогда равна:

$$L_{\text{CRF}}(y', y) = -\log \text{score}_{\text{CRF}}(x, y). \tag{19.12}$$

Это означает, что потеря связана с разностью между вероятностью правильной структуры и 1. Функцию потерь CRF можно рассматривать как обобщение перекрестной энтропии в задаче жесткой классификации на структурный случай.

Вычисление градиента потери в уравнении (19.12) так же сложно, как построение ассоциированного графа вычислений. Трудности вызывает знаменатель (*функция разбиения*), где требуется суммировать по структурам в $\mathcal{Y}$, количество которых может расти экспоненциально. Однако для некоторых задач существует алгоритм динамического программирования, который позволяет эффективно выполнить суммирование за полиномиальное время (алгоритм прямого-обратного хода Витерби для последовательностей и рекурсивный алгоритм максимизации правдоподобия Кока–Янгера–Касами для древовидных структур). Если такой алгоритм существует, то его можно также адаптировать для создания графа вычислений полиномиального размера.

19.1.4. Приближенный поиск

Иногда эффективного алгоритма поиска для задачи предсказания не существует. Может случиться, что отсутствует эффективный способ нахождения структуры с максимальной оценкой (решения задачи максимизации) в уравнениях (19.7), (19.8) или (19.9) или неизвестен эффективный алгоритм вычисления функции разбиения (знаменателя) в уравнении (19.11). В таких случаях можно прибегнуть к алгоритмам *приближенного вывода*, например лучевому поиску. При использовании лучевого поиска максимизация и суммирование производятся относительно элементов в луче. Например, лучевой поиск можно применить для поиска структуры с приближенно максимальной оценкой, а в случае функции разделения суммировать по структурам, оставшимся в луче, а не по всему экспоненциально большому множеству $\mathcal{Y}(x)$. К неточному поиску относится также техника *раннего обновления*: вместо того чтобы вычислять потерю по полным структурам, производить вычисление по частичным структурам, как только золотые элементы выпадают из луча. Анализ техники раннего обновления и альтернативных методов вычисления потери, а также стратегий обновления в случае, когда обучение проводится в условиях приближенного поиска, имеется в работе Huang et al. [2012].

19.1.5. Переранжирование

Если поиск по всем возможным структурам не осуществим, не эффективен или с трудом интегрируется в модель, то альтернативой лучевому поиску является *переранжирование*. В этом случае [Charniak and Johnson, 2005, Collins and Koo, 2005] базовая модель используется для порождения списка структур с k лучшими оценками. Затем обучается более сложная модель, которая призвана оценить кандидатов из списка k лучших так, чтобы структура, наилучшая относительно золотой, получила максимальную оценку. Поскольку теперь поиск производится по k элементам, а не по экспоненциально большому пространству, то сложная модель может быть обусловлена произвольными аспектами оцениваемой структуры (и будет выделять соответствующие признаки). Базовая модель, используемая для предсказания k лучших структур, может быть более простой, с более сильными предположениями о независимости, способной порождать разумные, но не превосходные результаты. Методы переранжирования являются естественными кандидатами для структурного предсказания с помощью нейронных сетей, поскольку позволяют модели сконцентрироваться на выделении признаков и структуре сети, устраняя необходимость интеграции механизма нейросетевой оценки с декодером. На самом деле методы переранжирования часто использу-

ются для экспериментов с нейросетевыми моделями, которые с трудом интегрируются с декодером, например со сверточными, рекуррентными и рекурсивными сетями. Изучению таких методов посвящены работы Auli et al. [2013], Le and Zuidema [2014], Schwenk et al. [2006], Socher et al. [2013a], Zhu et al. [2015a] и Choe and Charniak [2016].

19.1.6. Смотрите также

Помимо примеров из раздела 19.4, CRF для последовательностей с нейросетевыми потенциалами клики обсуждаются в работах Peng et al. [2009] и Do et al. [2010], где применяются к пометке последовательностей биологических данных, данных оптического распознавания символов и речевых сигналов, и в работе Wang and Manning [2013], где используются в традиционных задачах разметки текстов на естественном языке (разбиение на блоки и распознавание именованных сущностей). Похожая архитектура разметки последовательностей описана в работах Collobert and Weston [2008], Collobert et al. [2011]. Подход на основе кусочно-линейной функции потерь использовался в работе Pei et al. [2015] для разбора зависимостей методом разложения на дуги с экстрактором признаков, заданным вручную, и в работе Kiperwasser and Goldberg [2016b] с biLSTM в качестве экстрактора. Вероятностный подход применялся в работе Durrett and Klein [2015] для построения CRF-анализатора грамматики составляющих. Функция разбиения на основе приближенного лучевого поиска (приближенная CRF) с успехом использована для построения анализатора на основе переходов состояний в работе Zhou et al. [2015], а позднее для решения различных задач в работе Andor et al. [2016].

19.2. Жадное структурное предсказание

Помимо методов структурного предсказания на основе поиска, существуют жадные подходы, когда структурная задача разлагается в последовательность локальных задач предсказания и классификатор обучается хорошо принимать каждое локальное решение. На этапе тестирования обученный классификатор используется в жадном режиме. Примерами такого подхода могут служить модели разметки слева направо [Giménez and Màrquez, 2004] и жадный синтаксический разбор на основе переходов состояний [Nivre, 2008][1]. Поскольку поиск не предполагается, жадные подходы не ограничиваются доступными им видами признаков, а могут использовать развитые обусловливающие структуры. Благодаря этому жадные подходы вполне конкурентоспособны во многих задачах с точки зрения верности предсказаний.

Однако жадные подходы по определению эвристические и потенциально уязвимы для проблемы *распространения ошибки*: ошибки предсказания, сделанные в начале последовательности, уже невозможно исправить, и они могут накапливаться со временем. Эта проблема особенно серьезна, если используется метод с *ограниченным горизонтом* просмотра последовательности, как, например, оконные экстракторы признаков. Такие методы обрабатывают лексемы после-

[1] Анализаторы на основе переходов состояния выходят за рамки этой книги, но познакомиться с ними можно в работах Kübler et al. [2008], Nivre [2008] и Goldberg and Nivre [2013].

довательности в фиксированном порядке и видят только локальное окно вокруг точки предсказания. Они ничего не могут знать о будущем последовательности и с большой вероятностью будут уведены локальным контекстом в неверном направлении.

По счастью, использование РНС (и особенно biRNN) значительно сглаживает этот эффект. Экстрактор признаков на основе biRNN может просмотреть все входные данные до конца, и его можно обучить выделять полезные признаки из сколько угодно далеко находящихся позиций. Эта особенность превращает жадные локальные модели, обученные с применением biRNN-экстрактора, в жадные *глобальные* модели: каждое решение можно обусловить всей последовательностью, что делает процесс менее склонным к последующему «удивлению» при виде неожиданного выхода. Поскольку каждое предсказание становится более верным, общая верность также заметно повышается.

На самом деле работы по синтаксическому анализу показывают, что модели жадного предсказания, обученные с помощью глобальных biRNN-экстракторов признаков, могут соперничать в верности с методами на основе поиска, которые сочетают глобальный поиск с локальными экстракторами признаков [Cross and Huang, 2016a, Dyer et al., 2015, Kiperwasser and Goldberg, 2016b, Lewis et al., 2016, Vaswani et al., 2016].

В дополнение к глобальным экстракторам признаков жадные методы можно улучшить, применяя технику обучения, направленную на сглаживание проблемы распространения ошибки, – либо за счет попытки делать сначала легкие предсказания, а только потом трудные (подход «сначала легкие» из работы [Goldberg and Elhadad, 2010]), либо за счет того, что условия обучения делаются более похожими на условия тестирования путем предъявления процедуре обучения входов, вероятно, содержащих ошибки [Hal Daumé III et al., 2009, Goldberg and Nivre, 2013]. Это эффективно также для обучения жадных нейросетевых моделей, как показано в работах Ma et al. [2014] (разметчик на основе принципа «сначала легкие») и Ballesteros et al. [2016], Kiperwasser and Goldberg [2016b] (обучение динамического оракула для жадного разбора зависимостей).

19.3. Условная генерация как предсказание структурного выхода

Наконец, РНС-генераторы, особенно в системах условной генерации (глава 17), также можно рассматривать как примеры структурного предсказания. Серия предсказаний, сделанных генератором, порождает структурный выход $\hat{t}_{1:n}$. С каждым отдельным предсказанием ассоциирована оценка (или вероятность) $score(\hat{t}_i | \hat{t}_{i-1})$, а нас интересует выходная последовательность с максимальной оценкой (или максимальной вероятностью), т. е. такая, для которой сумма $\sum_{i=1}^{n} score(\hat{t}_i | \hat{t}_{i-1})$ достигает максимума. К сожалению, из-за немарковской природы РНС функцию оценки *нельзя* разложить на факторы, которые позволили бы осуществить точный поиск стандартными методами динамического программирования, и приходится использовать приближенный поиск.

В одном из популярных приближенных методов используется *жадное предсказание*, когда на каждом этапе берется элемент с максимальной оценкой. Хотя этот подход часто оказывается эффективным, очевидно, что он не оптимален. Дей-

ствительно, применение в качестве приближения лучевого поиска часто работает намного лучше жадного подхода.

На этом этапе важно принять во внимание, как *обучаются* условные генераторы. Как описано в разделе 17.1.1, генераторы обучаются с применением техники *принуждения со стороны учителя*: используется вероятностная целевая функция, которая пытается назначить большую массу вероятности золотым наблюдавшимся последовательностям. Если дана золотая последовательность $t_{1:n}$, то на каждом этапе i модель обучается назначать большую массу вероятности золотому событию $\hat{t}_i = t_i$, обусловленному золотой историей $t_{1:i-1}$.

У этого подхода два недостатка. Во-первых, он основан на *золотой истории* $t_{1:i-1}$, тогда как на практике перед генератором будет стоять задача назначать оценки в соответствии с *предсказанной историей* $\hat{t}_{1:i-1}$. Во-вторых, это локально нормированная модель: она назначает распределение вероятностей каждого события и потому подвержена проблеме *смещения метки*[1], которая может негативно сказаться на качестве решений, возвращаемых лучевым поиском. Обе проблемы получили внимание со стороны сообществ NLP и машинного обучения, но пока еще не полностью исследованы в контексте РНС-генерации.

Первую проблему можно смягчить, используя такие протоколы обучения, как SEARN [Hal Daumé III et al., 2009], DAGGER [Ross and Bagnell, 2010, Ross et al., 2011] и разведывательное обучение с помощью динамических оракулов [Goldberg and Nivre, 2013]. Применение этих методов в контексте РНС-генераторов предложено в работе Bengio et al. [2015] под названием *плановая выборка* (scheduled sampling).

Для решения второй проблемы можно отказаться от локально нормированной целевой функции и перейти к глобальным целевым функциям на уровне последовательности, которые больше подходят для лучевого декодирования. К таковым относятся лучевые аппроксимации структурной кусочно-линейной функции потерь [уравнение (19.8)] и CRF-потеря [уравнение (19.11)], рассмотренная в разделе 19.1.4 выше. В работе Wiseman and Rush [2016] обсуждаются глобальные (на уровне последовательности) функции оценки для РНС-генераторов.

19.4. Примеры

19.4.1. Структурное предсказание на основе поиска: анализ зависимостей первого порядка

Рассмотрим задачу анализа зависимостей, описанную в разделе 7.7. На вход подается предложение из n слов $\mathrm{s} = w_1, \ldots, w_n$, и требуется найти для него *дерево разбора зависимостей* y (рис. 7.1). Это ориентированное дерево с корнем, построенное по словам предложения. Каждому слову в дереве соответствует единственный родитель, который может быть либо другим словом предложения, либо специальным элементом ROOT. Родительское слово называется *заглавным*, а его дочерние слова – *модификаторами*.

Анализ зависимостей прекрасно ложится на систему структурного предсказания на основе поиска, описанную в разделе 19.1. Точнее, уравнение (19.5) говорит,

[1] Обсуждение проблемы смещения метки *см.* в разделе 3 статьи Andor et al. [2016] и в упоминаемых в ней работах.

что мы должны назначать оценки деревьям, разлагая их на части и оценивая каждую часть по отдельности. В литературе по синтаксическому анализу описывается много вариантов разложения [Koo and Collins, 2010, Zhang and McDonald, 2012], мы займемся простейшим, взятым из работы McDonald et al. [2005]: *разложением на дуги*. Каждая часть будет представлена дугой дерева (т. е. парой, состоящей из заглавного слова w_h и модификатора w_m). Каждая дуга (w_h, w_m) будет оцениваться отдельно с помощью локальной функции оценки качества присоединения. Назначив оценки каждой из n^2 возможных дуг, мы можем выполнить *алгоритм вывода*, например *алгоритм Эйснера* [Eisner and Satta, 1999, Kübler et al., 2008, McDonald et al., 2005], который находит допустимое проективное дерево[1] с максимальной суммой оценок дуг. Уравнение (19.5) тогда принимает вид:

$$\text{score}_{\text{global}}(x, y) = \sum_{(w_h, w_m) \in y} \text{score}_{\text{local}}(w_h, w_m) = \sum_{(w_h, w_m) \in y} \text{NN}(\phi(h, m, s)), \qquad (19.13)$$

где $\phi(h, m, s)$ – функция выделения признаков, преобразующая индексы слов предложения h и m в вещественные векторы. Мы обсуждали экстракторы признаков для задачи синтаксического анализа в разделах 7.7 и 8.6 (где использовались вручную спроектированные признаки) и в разделе 16.2.3 (с использованием biRNN-экстрактора). Итак, предположим, что экстрактор признаков задан, и обратимся к процедуре обучения.

Определившись с конкретной формой нейросетевого компонента (скажем, МСП, $\text{NN}(\boldsymbol{x}) = (\tanh(\boldsymbol{x}\boldsymbol{U} + \boldsymbol{b}))\cdot\boldsymbol{v}$), мы легко можем вычислить оценку $a_{[h,m]}$ каждой возможной дуги (в предположении, что индекс узла ROOT равен 0):

$$a_{[h,m]} = (\tanh(\phi(h, m, s))\boldsymbol{U} + \boldsymbol{b})\cdot\boldsymbol{v}, \qquad \begin{array}{l} \forall h \in 0, \ldots, n; \\ \forall m \in 1, \ldots, n. \end{array} \qquad (19.14)$$

Затем выполним алгоритм Эйснера, который дает предсказанное дерево y' с максимальной оценкой:

$$y' = \max_{y \in \mathcal{Y}} \sum_{(h,m) \in y} a_{[h,m]} = \text{Eisner}(n, \boldsymbol{a}).$$

Если бы мы использовали вывод, дополненный функцией стоимости, то взяли бы вместо этого оценки $\bar{\boldsymbol{a}}$:

$$\bar{a}_{[h,m]} = a_{[h,m]} + \begin{cases} 0 & \text{если } (h, m) \in y \\ \rho & \text{в противном случае} \end{cases}.$$

Имея предсказанное дерево y' и золотое дерево y, мы можем создать граф вычислений для структурной кусочно-линейной функции потерь:

---

[1] Специалисты по синтаксическому анализу говорят о *проективных* и *непроективных* деревьях. Проективность налагает дополнительные ограничения на форму дерева: оно должно быть построено по линейной последовательности слов предложения в исходном порядке, т. е. дуги не должны пересекаться с проекциями на слова исходного предложения. Хотя это различие важно для синтаксического анализа, оно выходит за рамки книги. Дополнительные сведения *см*. в работах Kübler et al. [2008] и Nivre [2008].

$$\max(0, 1 + \underbrace{\sum_{(h',m')\in y'} \tanh(\phi(h', m', s))\boldsymbol{U} + \boldsymbol{b})\cdot\boldsymbol{v}}_{\max_{y'\neq y}\text{score}_{\text{global}}(s,y')} - \underbrace{\sum_{(h,m)\in y} \tanh(\phi(h, m, s))\boldsymbol{U} + \boldsymbol{b})\cdot\boldsymbol{v}}_{\text{score}_{\text{global}}(s,y)}. \quad (19.15)$$

Затем вычисляем градиенты потери методом обратного распространения, обновляем параметры и переходим к следующему дереву в обучающем наборе.

Этот подход к синтаксическому анализу описан в работах Pei et al. [2015] (с использованием вручную спроектированного экстрактора признаков из раздела 8.6) и Kiperwasser and Goldberg [2016b] (с использованием biRNN-экстрактора из раздела 16.2.3).

19.4.2. Нейросетевые CRF для распознавания именованных сущностей

НЕЗАВИСИМАЯ КЛАССИФИКАЦИЯ Рассмотрим задачу распознавания именованных сущностей (NER), описанную в разделе 7.5. Эта задача сегментации последовательности часто моделируется как *разметка последовательности*: каждому слову предложения назначается одна из K BIO-меток, описанных в табл. 7.1, после чего решения о разметке детерминированно преобразуются в участки. В разделе 7.5 мы рассматривали задачу NER как задачу классификации слова в контексте, предполагая, что решения о метке каждого слова принимаются независимо.

В постановке с независимой классификацией дано предложение $s = w_1, \ldots, w_n$, и мы используем функцию выделения признаков $\phi(i, s)$ для создания вектора признаков, представляющего слово w_i в контексте предложения. Затем классификатор, например МСП, применяется для предсказания оценки (или вероятности) каждой метки:

$$\hat{\boldsymbol{t}}_i = \text{softmax}(\text{MLP}(\phi(i, s))), \quad \forall i \in 1, \ldots, n, \quad (19.16)$$

где $\hat{\boldsymbol{t}}_i$ – вектор предсказанных оценок меток, а $\hat{\boldsymbol{t}}_{i[k]}$ – оценка назначения метки k слову i. Тогда предсказанная разметка $\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_n$ предложения получается путем независимого выбора метки с наибольшей оценкой в каждой позиции предложения:

$$\hat{y}_i = \underset{k}{\text{argmax}}\, \hat{\boldsymbol{t}}_{i[k]}, \quad \forall i \in 1, \ldots, n, \quad (19.17)$$

а оценка разметки $\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_n$ равна:

$$\text{score}(s, \hat{\boldsymbol{y}}) = \sum_{i=1}^{n} \boldsymbol{t}_{i[\hat{y}_i]}. \quad (19.18)$$

СТРУКТУРНАЯ РАЗМЕТКА ПОСРЕДСТВОМ СОВМЕСТНЫХ РЕШЕНИЙ О ПАРАХ МЕТОК Независимая классификация хорошо работает во многих случаях, но она неоптимальна, потому что решения о разметке соседних слов могут влиять друг на друга. Рассмотрим, к примеру, последовательность *Paris Hilton*: первое слово может быть местоположением или персоной, а второе – организацией или персоной, но если одно из них признать персоной, то и второе должно быть признано

персоной с высокой долей уверенности. Мы хотели бы, чтобы решения о разметке могли влиять друг на друга, и отразить это в оценке. Чаще всего это достигается путем введения факторов метка–метка: оценок совместимости пар соседних меток. Интуитивно понятно, что пара B-PER I-PER должна получить высокую оценку, а пара B-PER I-ORG – очень низкую или даже отрицательную. Для набора K возможных меток введем матрицу оценивания $\boldsymbol{A} \in R^{K \times K}$, в которой элемент $\boldsymbol{A}_{[g,h]}$ – оценка совместимости последовательности меток g h.

Функция оценки для назначения меток модифицируется с учетом факторов метка–метка:

$$\text{score}(s, \hat{\mathbf{y}}) = \sum_{i=1}^{n} \boldsymbol{t}_{\boldsymbol{i}[\hat{y}_i]} + \sum_{i=1}^{n+1} \boldsymbol{A}_{[\hat{y}_{i-1}, \hat{y}_i]}, \tag{19.19}$$

где в позициях 0 и $n + 1$ находятся специальные символы *START* и *END*. Зная оценки меток отдельных слов $\boldsymbol{t}_{1:n}$ и значения элементов $\boldsymbol{A}$, можно найти последовательность $\hat{\mathbf{y}}$, доставляющую максимум выражению (19.19), применив алгоритм динамического программирования Витерби.

Поскольку не требуется, чтобы оценки всех меток были положительны и в сумме равны 1, мы устраняем функцию softmax из вычисления оценок $\boldsymbol{t}_i$:

$$\hat{\boldsymbol{t}}_i = \text{MLP}(\phi(i, s)), \quad \forall i \in 1, \ldots, n. \tag{19.20}$$

Оценки $\boldsymbol{t}_i$ вычисляются нейронной сетью в соответствии с уравнением (19.20), а матрицу $\boldsymbol{A}$ можно рассматривать как дополнительные параметры модели. Теперь можно обучить структурную модель со структурной кусочно-линейной потерей [уравнение (19.8)] или структурной кусочно-линейной потерей, дополненной функцией стоимости [уравнение (19.9)].

Но мы вместо этого последуем работе Lample et al. [2016] и воспользуемся вероятностной целевой функцией CRF.

СТРУКТУРНОЕ ОБУЧЕНИЕ С ЦЕЛЕВОЙ ФУНКЦИЕЙ CRF Если используется целевая функция CRF, то наша цель – назначить вероятность каждой возможной последовательности меток $\mathbf{y} = y_1, \ldots, y_n$ для слов предложения s. Это моделируется посредством вычисления softmax от всех возможных разметок:

$$\begin{aligned} \text{score}_{\text{CRF}}(s, \mathbf{y}) = P(\mathbf{y} \mid s) &= \frac{e^{\text{score}(s, \mathbf{y})}}{\sum_{y' \in \mathcal{Y}(s)} e^{\text{score}(s, \mathbf{y}')}} \\ &= \frac{\exp\left(\sum_{i=1}^{n} \boldsymbol{t}_{\boldsymbol{i}[y_i]} + \sum_{i=1}^{n} \boldsymbol{A}_{[y_i, y_{i-1}]}\right)}{\sum_{\mathbf{y}' \in \mathcal{Y}(s)} \exp\left(\sum_{i=1}^{n} \boldsymbol{t}_{\boldsymbol{i}[y'_i]} + \sum_{i=1}^{n} \boldsymbol{A}_{[y'_i, y'_{i-1}]}\right)}. \end{aligned} \tag{19.21}$$

Знаменатель один и тот же для всех разметок $\mathbf{y}$, поэтому нахождение наилучшей последовательности (без определения ее вероятности) сводится к нахождению последовательности, доставляющей максимум функции $\text{score}(s, y)$, и это можно сделать с помощью алгоритма Витерби, как описано выше.

Потеря определяется как отрицательное логарифмическое правдоподобие правильной структуры y:

$$
\begin{aligned}
-\log P(\mathbf{y}\mid s) &= -\left(\sum_{i=1}^{n+1}\boldsymbol{t}_{i[y_i]}+\sum_{i=1}^{n+1}\boldsymbol{A}_{[y_{i-1},\,y_i]}\right)+\log\sum_{\mathbf{y}'\in\mathcal{Y}(s)}\exp\left(\sum_{i=1}^{n+1}\boldsymbol{t}_{i[y'_i]}+\sum_{i=1}^{n+1}\boldsymbol{A}_{[y'_{i-1},\,y'_i]}\right)\\
&= \underbrace{-\left(\sum_{i=1}^{n+1}\boldsymbol{t}_{i[y_i]}+\sum_{i=1}^{n+1}\boldsymbol{A}_{[y_{i-1},\,y_i]}\right)}_{\text{оценка золотой структуры}}+\underbrace{\bigoplus_{\mathbf{y}'\in\mathcal{Y}(s)}\left(\sum_{i=1}^{n+1}\boldsymbol{t}_{i[y'_i]}+\sum_{i=1}^{n+1}\boldsymbol{A}_{[y'_{i-1},\,y'_i]}\right)}_{\text{использовать динамическое программирование}},
\end{aligned}
\tag{19.22}
$$

где $\oplus$ означает сложение в логарифмическом пространстве (логарифмическое сложение), т. е. $\oplus(a,b,c,d)=\log(e^a+e^b+e^c+e^d)$. Первый член легко построить как граф вычислений, но со вторым возни больше, потому что требуется суммировать по n^k последовательностям, принадлежащим $\mathcal{Y}(s)$. По счастью, это можно сделать, воспользовавшись вариантом алгоритма Витерби[1], который мы опишем ниже.

СВОЙСТВА ЛОГАРИФМИЧЕСКОГО СЛОЖЕНИЯ Операция логарифмического сложения выполняет сложение в логарифмическом пространстве. Она обладает следующими свойствами, которые используются при разработке динамической программы. Все они доказываются тривиально, оставляем это читателю в качестве упражнения.

$$\bigoplus(a,b)=\bigoplus(b,a) \qquad \textbf{Коммутативность} \tag{19.23}$$

$$\bigoplus(a,\bigoplus(b,c))=\bigoplus(a,b,c) \qquad \textbf{Ассоциативность} \tag{19.24}$$

$$\bigoplus(a+c,b+c))=\bigoplus(a+b)+c \qquad \textbf{Дистрибутивность} \tag{19.25}$$

Обозначим $\mathcal{Y}(s,r,k)$ – множество последовательностей длины r, которые оканчиваются символом k. Тогда множество всех возможных последовательностей над $|s|$ можно записать в виде $\mathcal{Y}(s)=\mathcal{Y}(s,n+1,\text{*END*})$. Обозначим далее $\mathcal{Y}(s,r,\ell,k)$ множество последовательностей длины r, в которых последним символом является k, а предпоследним – ℓ. Положим, $\Gamma[r,k]=\bigoplus_{\mathbf{y}'\in\mathcal{Y}(s,r,k)}\sum_{i=1}^{r}(\boldsymbol{t}_{i[y'_i]}+\boldsymbol{A}_{[y'_{i-1},y'_i]})$. Наша цель – вычислить $\Gamma[n+1,\text{*END*}]$. Для краткости будем писать $f(i,y'_{i-1},y'_i)=\boldsymbol{t}_{i[y'_i]}+\boldsymbol{A}_{[y'_{i-1},y'_i]}$. Теперь получаем:

$$
\begin{aligned}
\Gamma[r,k] &= \bigoplus_{\mathbf{y}'\in\mathcal{Y}(s,r,k)}\sum_{i=1}^{r}f(i,y'_{i-1},y'_i);\\
\Gamma[r+1,k] &= \bigoplus_{\ell}\bigoplus_{\mathbf{y}'\in\mathcal{Y}(s,r+1,\ell,k)}\left(\sum_{i=1}^{r+1}f(i,y'_{i-1},y'_i)\right)\\
&= \bigoplus_{\ell}\bigoplus_{\mathbf{y}'\in\mathcal{Y}(s,r+1,\ell,k)}\left(\sum_{i=1}^{r}(f(i,y'_{i-1},y'_i))+f(r+1,y'_{r-1}=\ell,y'_r=k)\right)\\
&= \bigoplus_{\ell}\left(\bigoplus_{\mathbf{y}'\in\mathcal{Y}(s,r+1,\ell,k)}\left(\sum_{i=1}^{r}f(i,y'_{i-1},y'_i)\right)+f(r+1,y'_{r-1}=\ell,y'_r=k)\right)\\
&= \bigoplus_{\ell}\big(\Gamma[r,\ell]+f(r+1,y'_{r-1}=\ell,y'_r=k)\big)\\
&= \bigoplus_{\ell}\big(\Gamma[r,\ell]+\boldsymbol{t}_{r+1[k]}+\boldsymbol{A}_{[\ell,k]}\big).
\end{aligned}
$$

[1] Этот алгоритм называется *прямым*, но это не то же самое, что алгоритм для вычислений на *прямом* проходе по графу вычислений.

Мы получили рекуррентное соотношение:

$$\Gamma[r+1,k]=\bigoplus_{\ell}\left(\Gamma[r,\ell]+\boldsymbol{t}_{r+1[k]}+\boldsymbol{A}_{[\ell,k]}\right), \tag{19.26}$$

которое можно использовать для построения графа вычисления знаменателя $\Gamma[n+1, \text{*END*}]$[1]. Построив граф, мы сможем вычислить градиенты методом обратного распространения.

19.4.3. Аппроксимация CRF в задаче NER лучевым поиском

В предыдущем разделе мы преобразовали задачу распознавания именованных сущностей (NER) в структурную задачу, поместив совместные решения о выходных метках в позициях i и $i-1$ в матрицу оценивания $\boldsymbol{A}$, которая содержит оценки для каждой пары соседних меток. Это напоминает использование марковского предположения первого порядка, согласно которому метка в позиции i зависит только от метки в позиции $i-1$ и не зависит от более ранних. Предположение о независимости позволило нам разложить оценку последовательности и построить эффективные алгоритмы нахождения последовательности с наибольшей оценкой, а также осуществить суммирование по всем возможным последовательностям меток.

Иногда желательно ослабить марковское предположение о независимости и вместо этого обусловливать метку y_i всеми предыдущими метками $y_{1:i-1}$. Это можно включить в модель разметки, добавив РНС по истории меток. Теперь мы оцениваем последовательность меток $\mathbf{y}=y_1,\ldots,y_n$ следующим образом:

$$\text{score}(s,\hat{\mathbf{y}})=\sum_{i=1}^{n+1} f([\phi(s,i);RNN(\hat{\mathbf{y}}_{1:i})]), \tag{19.27}$$

где f – параметрическая функция, например линейное преобразование или МСП, а ϕ – функция выделения признаков, отображающая слово в позиции i предложения s в вектор[2]. Проще говоря, мы вычисляем локальную оценку назначения метки k слову в i-й позиции предложения, рассматривая признаки в i-й позиции, а также РНС, кодирующую последовательность меток y_1, y_2, y_{i-1}, k. Затем мы вычисляем глобальную оценку как сумму локальных оценок.

К сожалению, РНС увязывает локальные оценки по всем предыдущим решениям о разметке, что не позволяет нам использовать эффективные алгоритмы динамического программирования для точного нахождения наилучшей разметки последовательности или суммирования по всем допускаемым моделью последовательностям меток. Вместо этого придется обратиться к аппроксимации, например *лучевому поиску*. С помощью луча размера r мы сможем найти r различных последовательностей меток $\hat{\mathbf{y}}_1,\ldots,\hat{\mathbf{y}}_r$[3]. В качестве приблизительно лучшей по-

---

[1] Заметим, что это то же самое рекуррентное соотношение, что в алгоритме поиска наилучшего пути Витерби, только место max заняла операция $\oplus$.

[2] Функция выделения признаков ϕ может быть основана на окне слов или на biLSTM-сети, как и функции для частеречной разметки из разделов 8.5 и 16.2.1.

[3] Алгоритм лучевого поиска работает поэтапно. Получив r возможных последовательностей меток длины i ($\hat{\mathbf{y}}^1_{1:i},\ldots,\hat{\mathbf{y}}^r_{1:i}$), соответствующих первым i словам последовательности, мы дополняем каждую последовательность всеми возможными метками, оцениваем каждую из получившихся $r\times K$ последовательностей и сохраняем r последовательностей с наибольшими оценками. Процесс продолжается, пока не будет получено r последовательностей меток для всего предложения.

следовательности меток берется та из r лучевых последовательностей, у которой оценка максимальна:

$$\underset{i\in 1,\ldots,r}{\operatorname{argmax}} \operatorname{score}(s, \hat{\mathbf{y}}^i).$$

Для обучения можно использовать следующую целевую функцию, аппроксимирующую CRF:

$$\operatorname{score}_{\text{ApproxCrf}}(s, y) = \tilde{P}(\mathbf{y} \mid s) = \frac{e^{\operatorname{score}(s,\mathbf{y})}}{\sum_{\mathbf{y}'\in\tilde{\mathcal{Y}}(s,r)} e^{\operatorname{score}(s,\mathbf{y}')}}; \tag{19.28}$$

$$\begin{aligned} L_{\text{CRF}}(y', y) &= -\log \tilde{P}(\mathbf{y} \mid s) \\ &= -\operatorname{score}(s, \mathbf{y}) + \log \sum_{\mathbf{y}'\in\tilde{\mathcal{Y}}(s,r)} e^{\operatorname{score}(s,\mathbf{y}')}; \\ \tilde{\mathcal{Y}}(s, r) &= \{\mathbf{y}^1, \ldots, \mathbf{y}^r\} \cup \{\mathbf{y}\}. \end{aligned} \tag{19.29}$$

Вместо нормировки посредством суммирования по всему множеству последовательностей $\mathcal{Y}(s)$ мы суммируем по $\tilde{\mathcal{Y}}(s, r)$ – объединению золотой и r лучевых последовательностей. Число r невелико, поэтому суммирование тривиально. Когда r стремится к n^K, наша аппроксимация стремится к истинной целевой функции CRF.

Глава 20

Обучение каскадное, многозадачное и с частичным привлечением учителя

При обработке естественного языка часто бывает, что несколько задач передают данные друг другу. Например, синтаксический анализатор, который мы обсуждали в разделах 7.7, 16.2.3 и 19.4.1, принимает метки частей речи, которые сами автоматически предсказываются статистической моделью. Система, в которой одна модель подает предсказания на вход другой, независимой от нее, называется *конвейером*. Альтернативный подход – *каскадирование моделей*. В этом случае мы вместо подачи *предсказаний* модели A (разметчика) на вход модели B (анализатора) подаем анализатору *промежуточные представления*, информативные для предсказания меток. То есть вместо того чтобы принять определенное решение о разметке, мы возлагаем разрешение неопределенности на анализатор. В системе глубокого обучения реализовать каскадирование очень просто, нужно лишь передать вектор, имеющий место до вызова argmax, или даже один из скрытых векторов.

Родственная техника называется *многозадачным обучением* [Caruana, 1997]; в этом случае мы имеем несколько взаимосвязанных задач предсказания (которые могут передавать данные друг другу, а могут и не передавать) и хотели бы воспользоваться информацией одной из задач для повышения верности предсказания в других. В глубоком обучении мы заводим разные сети для разных задач, но разрешаем им частично *разделять* структуру и параметры. Таким образом, образуется общее прогностическое ядро (разделяемая структура), на которое оказывают влияние все задачи, и обучающие данные для одной задачи могут улучшить предсказания других.

Каскадный подход естественно адаптируется к каркасу многозадачного обучения: вместо того чтобы просто передавать промежуточный выход разметчика анализатору, мы можем подставить в граф вычислений подграф, который будет отвечать за промежуточное представление разметки, и производить обратное распространение ошибки анализатора по всему пути до базового компонента разметки (который теперь стал разделяемым).

Еще один родственный и похожий случай – *обучение с частичным привлечением учителя*, когда мы имеем аннотированные учителем обучающие данные для

задачи A и хотим использовать аннотированные или неаннотированные данные для других задач, чтобы улучшить качество предсказания в задаче A.

Эти три метода мы и рассмотрим в настоящей главе.

20.1. Каскадирование моделей

При каскадировании моделей большие сети строятся из меньших компонентов. Например, в разделе 16.2.1 мы описали рекуррентную нейронную сеть для предсказания части речи слова на основе его контекста в предложении и составляющих его литер. В конвейере такую сеть можно было бы использовать как частеречный разметчик, а выработанные им предсказания подавать в качестве входных признаков нейронной сети, которая осуществляет разбиение на синтаксические блоки или производит синтаксический анализ.

А можно интерпретировать скрытые слои сети как кодировку, улавливающую информацию, релевантную предсказанию частей речи. Каскадный подход позволяет взять скрытые слои сети и соединить их (а не сами предсказания частей речи) с входным слоем сети синтаксического анализа. Тогда у нас появится более крупная сеть, которая принимает входные последовательности слов и литер, а выводит синтаксическую структуру.

В качестве конкретного примера рассмотрим сети для разметки и синтаксического анализа, описанные в разделах 16.2.1 и 16.2.3. Сеть разметки [уравнение (16.4)], воспроизведенная ниже, предсказывает метку i-го слова:

$$\begin{aligned} t_i &= \arg\max_j \operatorname{softmax}(\text{MLP}(\text{biRNN}(\boldsymbol{x}_{1:n}, i)))_{[j]}; \\ \boldsymbol{x}_i &= \phi(s, i) = [\boldsymbol{E}_{[w_i]}; \text{RNN}^f(\boldsymbol{c}_{1:\ell}); \text{RNN}^b(\boldsymbol{c}_{\ell:1})], \end{aligned} \tag{20.1}$$

а сеть синтаксического анализа [уравнение (16.6)] назначает оценки дугам согласно уравнениям:

$$\begin{aligned} \text{ArcScore}(h, m, w_{1:n}, t_{1:n}) &= \text{MLP}(\phi(h, m, s)) = \text{MLP}([\boldsymbol{v}_h; \boldsymbol{v}_m]); \\ \boldsymbol{v}_{1:n} &= \text{biRNN}^*(\boldsymbol{x}_{1:n}); \\ \boldsymbol{x}_i &= [\boldsymbol{w}_i; \boldsymbol{t}_i]. \end{aligned} \tag{20.2}$$

Важно отметить, что анализатор принимает входные слова $w_{1:n}$ *и метки* $t_{1:n}$, а затем преобразует их в векторы погружения, которые конкатенирует, чтобы образовать представление входа $\boldsymbol{x}_{1:n}$.

При каскадном подходе мы передаем состояние разметчика до предсказания непосредственно анализатору, так что получается одна совместная сеть. Конкретно обозначим $\boldsymbol{z}_i$ промежуточное состояние разметчика для i-го слова: $\boldsymbol{z}_i = \text{MLP}(\text{biRNN}(\boldsymbol{x}_{1:n}, i))$. Теперь можно использовать $\boldsymbol{z}_i$ в качестве входного представления i-го слова в анализаторе, т. е.

$$\begin{aligned} \text{ArcScore}(h, m, w_{1:n}) &= \text{MLP}_{\text{parser}}(\phi(h, m, s)) = \text{MLP}_{\text{parser}}([\boldsymbol{v}_h; \boldsymbol{v}_m]); \\ \boldsymbol{v}_{1:n} &= \text{biRNN}^*_{\text{parser}}(\boldsymbol{z}_{1:n}); \\ \boldsymbol{z}_i &= \text{MLP}_{\text{tagger}}(\text{biRNN}_{\text{tagger}}(\boldsymbol{x}_{1:n}, i)); \\ \boldsymbol{x}_i &= \phi_{\text{tagger}}(s, i) = [\boldsymbol{E}_{[w_i]}; \text{RNN}^f_{\text{tagger}}(\boldsymbol{c}_{1:\ell}); \text{RNN}^b_{\text{tagger}}(\boldsymbol{c}_{\ell:1})]. \end{aligned} \tag{20.3}$$

Абстракция графа вычислений позволяет распространять градиенты ошибки функции потерь с уровня синтаксической сети на уровень литер[1].

На рис. 20.1 схематично показана вся сеть целиком.

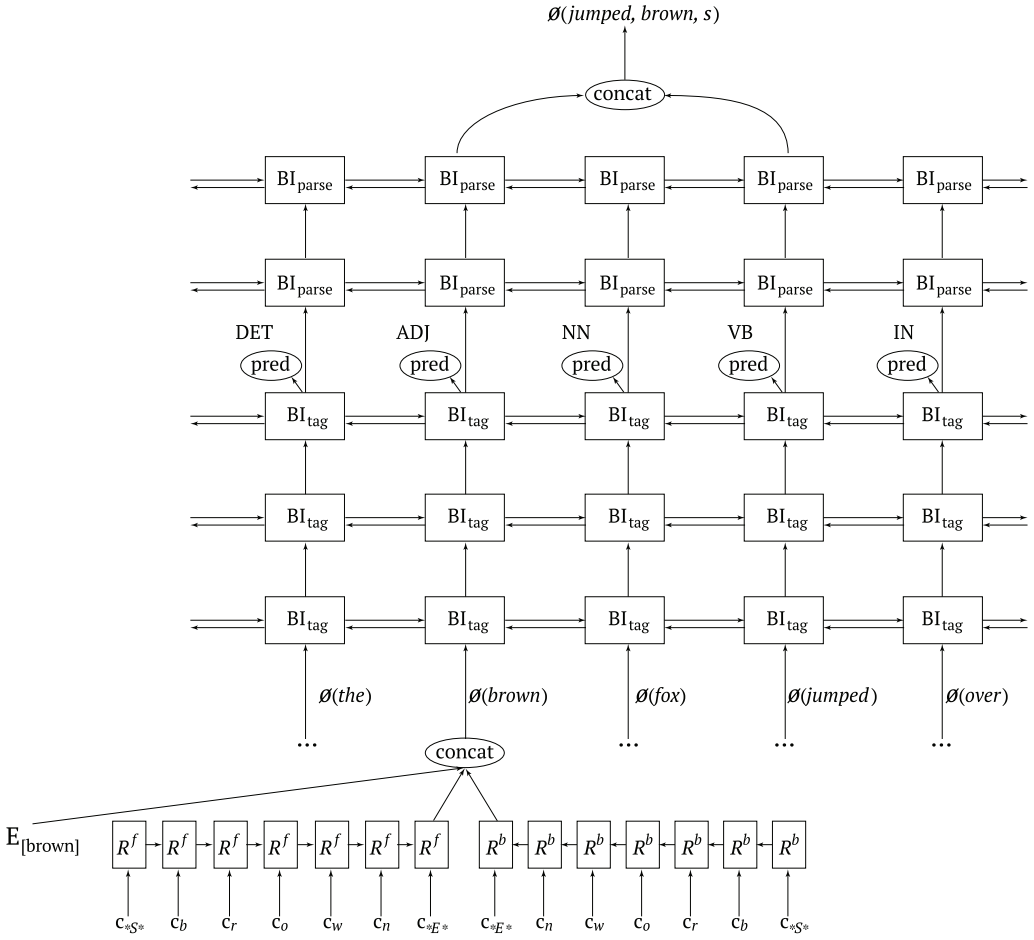


Рис. 20.1 ❖ Каскадная сеть разметки и синтаксического анализа [уравнение (20.3)]

Хотя у анализатора имеется доступ к самим словам, они уже могут быть размыты к моменту, когда пройдут все слои PHC-разметчика. Чтобы исправить ситуацию, можно использовать прямые связи, т. е. передавать погружения слов $\boldsymbol{E}_{[wi]}$ напрямую анализатору, а не только на выход разметчика:

[1] Действительно ли мы хотим производить обратное распространение ошибки по всему пути или нет, зависит от ситуации.

$$
\begin{aligned}
\text{ArcScore}(h, m, w_{1:n}) &= \text{MLP}_{\text{parser}}(\phi(h, m, s)) = \text{MLP}_{\text{parser}}([\boldsymbol{v}_h; \boldsymbol{v}_m]);\\
\boldsymbol{v}_{1:n} &= \text{biRNN}^{*}_{\text{parser}}(\boldsymbol{z}_{1:n});\\
\boldsymbol{z}_i &= [\boldsymbol{E}_{[w_i]}; \boldsymbol{z}'_i];\\
\boldsymbol{z}'_i &= \text{MLP}_{\text{tagger}}(\text{biRNN}_{\text{tagger}}(\boldsymbol{x}_{1:n}, i));\\
\boldsymbol{x}_i &= \phi_{\text{tagger}}(s, i) = [\boldsymbol{E}_{[w_i]}; \text{RNN}^{f}_{\text{tagger}}(\boldsymbol{c}_{1:\ell}); \text{RNN}^{b}_{\text{tagger}}(\boldsymbol{c}_{\ell:1})].
\end{aligned}
\tag{20.4}
$$

Эта архитектура изображена на рис. 20.2.

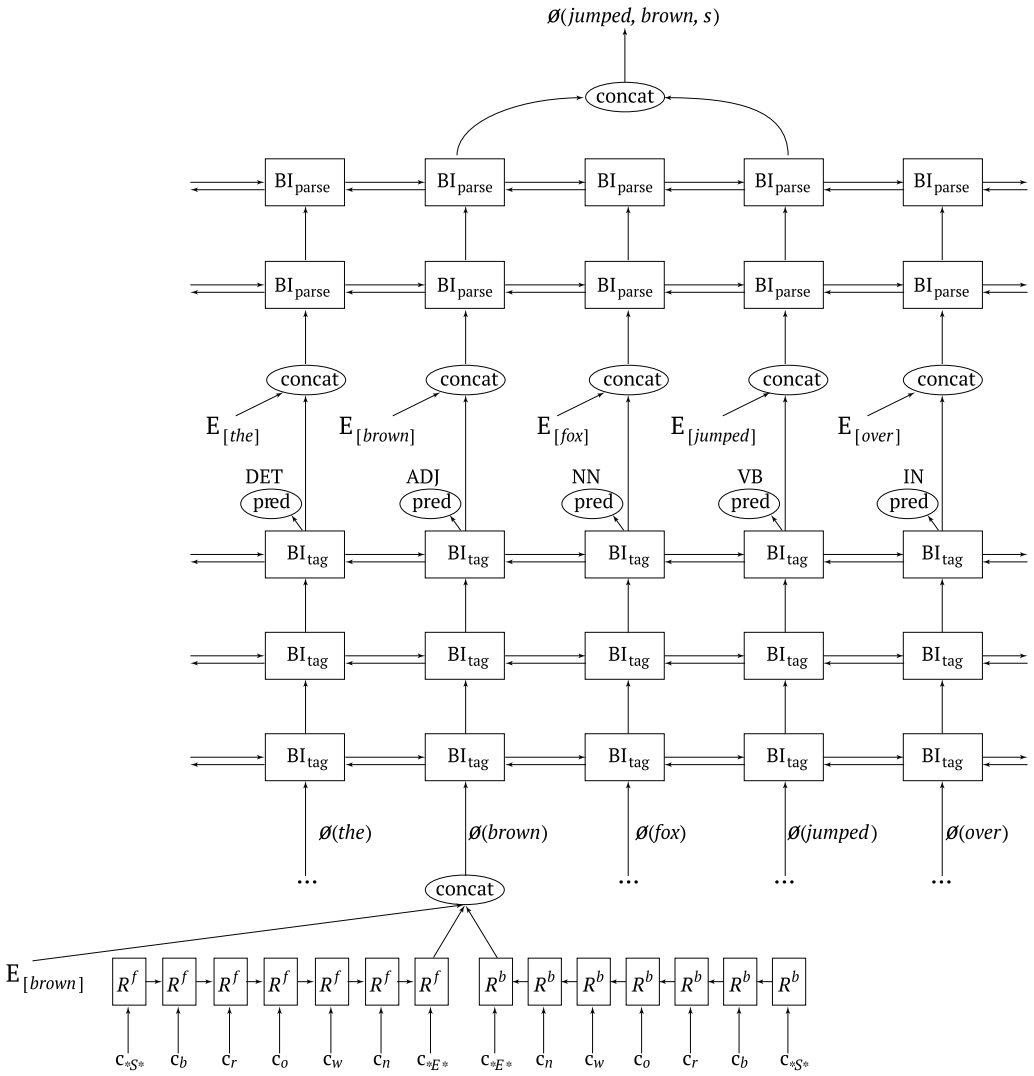


Рис. 20.2 ❖ Каскадная сеть разметки и синтаксического анализа с прямыми соединениями для погружений слов [уравнение (20.4)]

Для борьбы с проблемой исчезающего градиента в глубоких сетях, а также чтобы лучше использовать доступный обучающий материал, параметры отдельных

компонентов сети можно предварительно обучить на подходящей задаче, а только потом подавать в объемлющую сеть для дальнейшей настройки. Например, сеть предсказания частей речи можно обучить на сравнительно большом аннотированном корпусе, прежде чем подключать ее скрытый слой к сети синтаксического анализа, для которой доступно меньше обучающих данных. Если обучающие данные содержат сигналы учителя для обеих задач, то их можно использовать на этапе обучения, создав сеть с двумя выходами, по одному для каждой задачи, вычислять для каждого выхода свою потерю, а затем свести потери в один узел, из которого распространять градиенты ошибок.

Каскадирование моделей очень часто применяется при работе со сверточными, рекуррентными и рекурсивными сетями, когда, например, рекуррентная сеть используется для кодирования предложения вектором фиксированного размера, который затем подается на вход другой сети. Сигнал учителя для рекуррентной сети поступает в основном от следующей за ней сети, для которой выход рекуррентной сети является входом.

В нашем примере и разметчик, и анализатор основаны на опорной biRNN-сети. Это необязательно – одна или обе сети могли бы быть сетью прямого распространения, получающей на входе окно слов, сверточной сетью или любой другой архитектурой, которая порождает векторы и может передавать градиенты.

20.2. Многозадачное обучение

Многозадачное обучение (multi-task learning – MTL) – это родственная техника, применяемая, когда имеется несколько взаимосвязанных задач, которые мы считаем коррелированными в том смысле, что, обучившись решать одну, мы, вероятно, сможем получить «интуитивные соображения» о решении другой. Рассмотрим, к примеру, задачу *разбиения на синтаксические блоки* (*см*. врезку *Лингвистическое аннотирование* в разделе 6.2.2), в которой мы аннотируем предложение границами блоков, порождая такой вывод:

$[_{\text{NP}}$ the boy $]$ $[_{\text{PP}}$ with $]$ $[_{\text{NP}}$ the black shirt $]$ $[_{\text{VP}}$ opened $]$ $[_{\text{NP}}$ the door $]$ $[_{\text{PP}}$ with $]$ $[_{\text{NP}}$ a key $]$.

Как и распознавание именованных сущностей, разбиение на блоки – задача сегментации последовательности, и ее можно свести в задаче разметки с помощью схемы BIO-кодирования (*см*. раздел 7.5). Тогда сеть для разбиения можно смоделировать как глубокую biRNN, за которой следует МСП для предсказания отдельных меток:

$$\begin{aligned} p(\text{chunkTag}_i = j) &= \text{softmax}(\text{MLP}_{\text{chunk}}(\text{biRNN}_{\text{chunk}}(\boldsymbol{x}_{1:n}, i)))_{[j]}; \\ \boldsymbol{x}_i &= \phi(s, i) = \boldsymbol{E}^{\text{cnk}}_{[w_i]} \end{aligned} \tag{20.5}$$

(для краткости мы убрали из входа РНС уровня литер, но их очень просто добавить). Отметим, что это очень похоже на сеть частеречной разметки:

$$\begin{aligned} p(\text{posTag}_i = j) &= \text{softmax}(\text{MLP}_{\text{tag}}(\text{biRNN}_{\text{tag}}(\boldsymbol{x}_{1:n}, i)))_{[j]}; \\ \boldsymbol{x}_i &= \phi(s, i) = \boldsymbol{E}^{\text{tag}}_{[w_i]}. \end{aligned} \tag{20.6}$$

Обе сети изображены на рис. 20.3. Различными цветами обозначены разные наборы параметров.

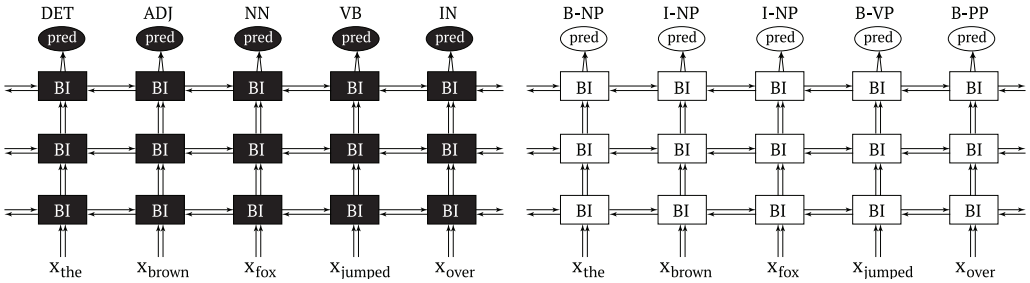


Рис. 20.3 ❖ Слева: сеть частеречной разметки. Справа: сеть разбиения на блоки

Между задачами частеречной разметки и разбиения на блоки имеется синергия. Предсказание границ блоков и частей речи опирается на некоторое общее базовое синтаксическое представление. Вместо того чтобы для каждой задачи обучать отдельную сеть, мы можем создать одну сеть с несколькими выходами. Тогда можно было бы обобществить параметры biRNN, но иметь специальные МСП-предсказатели для каждой задачи (или обобществить и МСП тоже, а специализировать для задачи только финальную матрицу и члены смещения). В результате получается такая разделяемая сеть:

$$\begin{aligned} p(\text{chunkTag}_i = j) &= \text{softmax}(\text{MLP}_{\text{chunk}}(\text{biRNN}_{\text{chunk}}(\boldsymbol{x}_{1:n}, i)))_{[j]}; \\ p(\text{posTag}_i = j) &= \text{softmax}(\text{MLP}_{\text{tag}}(\text{biRNN}_{\text{shared}}(\boldsymbol{x}_{1:n}, i)))_{[j]}; \\ \boldsymbol{x}_i &= \phi(s, i) = \boldsymbol{E}^{\text{shared}}_{[w_i]}. \end{aligned} \tag{20.7}$$

В обеих сетях используются одна и та же глубокая biRNN и слои погружения, но разные финальные выходные предикторы. Эта архитектура показана на рис. 20.4.

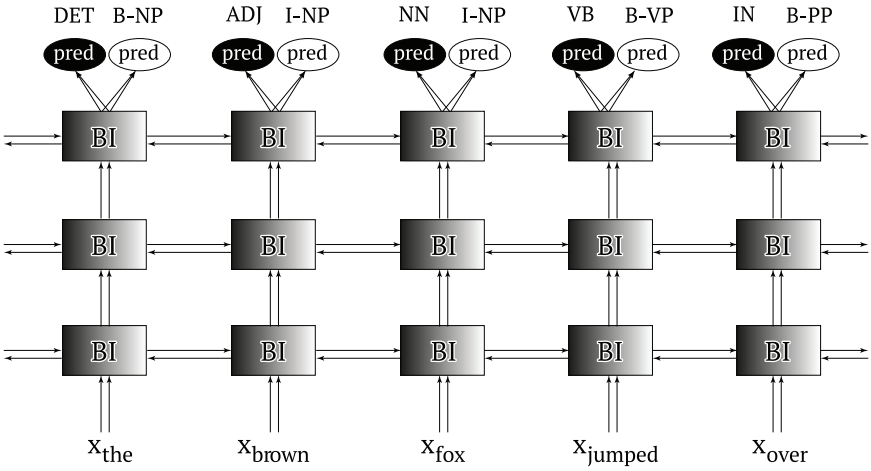


Рис. 20.4 ❖ Совместная сеть частеречной разметки и разбиения на блоки. Параметры biRNN общие, а сам компонент biRNN специализирован для каждой задачи. Финальные предикторы разделены

Большая часть параметров сети общая для разных задач. Поэтому полезная информация, полученная при обучении одной задачи, может помочь устранить неоднозначность в других.

20.2.1. Обучение в многозадачной конфигурации

Абстракция графа вычислений позволяет очень легко строить такие сети и вычислять для них градиенты – для этого нужно вычислить отдельную потерю для каждого доступного сигнала учителя, а затем просуммировать потери и свести их в один узел, который будет использоваться для вычисления градиентов. Если имеется несколько корпусов, каждый со своим сигналом учителя (например, один корпус для частеречной разметки, а другой – для разбиения на блоки), то предпочтителен такой протокол обучения: случайным образом выбрать корпус, прогнать пример через релевантную часть графа вычислений, вычислить потерю, выполнить обратное распространение ошибки и обновить параметры. На следующем шаге снова выбрать корпус случайным образом и т. д. На практике часто перетасовывают все доступные обучающие примеры и обрабатывают их по порядку. Важный момент заключается в том, что потенциально мы вычисляем градиенты разных функций потерь (и используем разные подсети) для каждого обучающего примера.

В некоторых случаях у нас может быть несколько задач, но наибольший интерес представляет одна из них. То есть имеется одна или более главных задач и несколько вспомогательных, которые, как мы полагаем, могут помочь решению главной задачи, но предсказания которых нас не интересуют. Тогда можно масштабировать потерю вспомогательной задачи, чтобы она была меньше потери главной. Другой вариант – сначала предобучить сеть на вспомогательных задачах, а затем взять общие компоненты этой сети и продолжить обучение только на главной задаче.

20.2.2. Избирательное обобществление

Возвращаясь к примеру сети для частеречной разметки и разбиения на блоки, мы могли бы возразить, что хотя обе задачи действительно разделяют информацию, задача частеречной разметки более *низкоуровневая*, чем разбиение на блоки: для разбиения на блоки необходима более детальная информация, чем для частеречной разметки. В таких случаях мы, возможно, предпочтем обобществлять между двумя задачами не всю глубокую biRNN-сеть, а лишь ее нижний уровень, специализировав верхние уровни для задачи разбиения на блоки (рис. 20.5).

Нижний уровень biRNN разделяется между двумя задачами. Основные сигналы учителя он получает от задачи частеречной разметки, но ему поступают и сигналы от задачи разбиения на блоки в виде градиентов. Верхние уровни занимаются только разбиением на блоки, но обучены хорошо работать с представлением, созданным нижними уровнями.

Этот подход к избирательному обобществлению заимствован из работы Søgaard and Goldberg [2016]. Похожий подход, в котором вместо рекуррентных сетей используются сети прямого распространения, описан в работе Zhang and Weiss [2016] под названием *стековое распространение* (stack propagation).

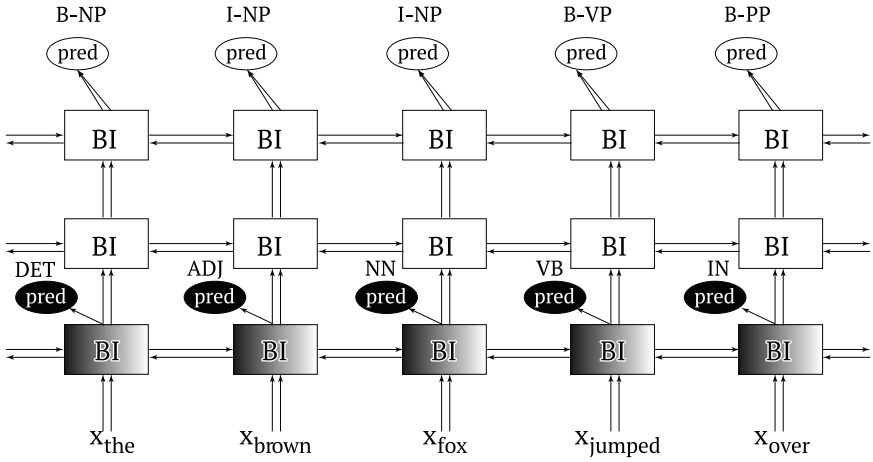


Рис. 20.5 ❖ *Избирательное обобществление* сетей частеречной разметки и разбиения на блоки. Нижний уровень biRNN разделяется обеими задачами, а верхние заняты только разбиением на блоки

Избирательно обобществленная сеть многозадачного обучения на рис. 20.5 по духу очень похожа на каскадную конфигурацию из предыдущего раздела (рис. 20.1). Действительно, очень трудно провести четкую границу между обоими подходами.

ИНВЕРСИЯ ВХОДА И ВЫХОДА Еще один взгляд на многозадачное и каскадное обучение связан с инверсией входа и выхода. Вместо того чтобы считать некоторый сигнал (скажем, метки частей речи) *входом* для задачи следующего уровня (скажем, синтаксического анализа), мы можем рассматривать его как *выходы* промежуточных слоев сети для задач более высокого уровня. То есть метки частей речи используются не как входы, а как сигнал учителя промежуточным слоям сети.

20.2.3. Предобучение погружений слов как многозадачное обучение

Задачи разбиения на блоки и частеречной разметки (да и многие другие) обладают синергией также с задачей *языкового моделирования*. Информация, необходимая для предсказания границы блока, – метка части речи слова – тесно связана с возможностью предсказать следующее или предыдущее слово: у задач общая синтактико-семантическая основа.

При таком взгляде использование предобученных векторов слов для инициализации слоя погружения предметно-ориентированной сети – это пример многозадачного обучения, в котором языковое моделирование является вспомогательной задачей. Алгоритмы погружения слов обучаются с дистрибутивной целевой функцией, которая обобщает языковое моделирование, а слой погружения слов этих алгоритмов затем *разделяется* с другой задачей.

Способ взаимодействия с учителем для алгоритма предобучения (т. е. выбор контекстов) должен соответствовать задаче, которую пытается решить специализированная сеть. Чем ближе задачи, тем больше выигрыш от многозадачного обучения.

20.2.4. Многозадачное обучение в условной генерации

MTL можно органично встроить в каркас условной генерации, рассмотренный в главе 17. Для этого нужен *разделяемый кодировщик*, подающий данные различным декодерам, каждый из которых пытается решать свою задачу. Это вынудит кодировщик кодировать информацию, релевантную всем задачам. Затем эту информацию можно не только обобществить между разными декодерами; потенциально она позволяет обучать разные декодеры на разных данных, увеличив тем самым общее количество примеров, доступных для обучения. Конкретный пример мы рассмотрим в разделе 20.4.4.

20.2.5. Многозадачное обучение как регуляризация

На многозадачное обучение можно также взглянуть как на *регуляризатор*. Сигналы учителя от вспомогательных задач препятствуют переобучению сети на главной задаче, вынуждая разделяемое представление быть более общим и полезным для предсказания не только на обучающих примерах для главной задачи. При таком взгляде на вещи, когда вспомогательные задачи предполагается использовать в качестве регуляризаторов, не следует выполнять MTL в последовательности, где сначала производится обучение вспомогательных задач, а потом представление адаптируется к главной задаче (как предлагалось в разделе 20.2.1). Вместо этого обучение всех задач следует вести параллельно.

20.2.6. Подводные камни

Хотя у многозадачного обучения весьма заманчивые перспективы, нельзя не упомянуть и о подводных камнях. MTL часто работает не слишком хорошо. Например, если задачи не тесно связаны, то MTL может не дать никакого выигрыша, а большинство задач как раз связано слабо. Выбор родственных задач для MTL – скорее искусство, чем наука.

Но даже если задачи тесно связаны, но емкость общей сети недостаточна для поддержки всех задач, качество каждой может ухудшиться. Если смотреть на MTL как на регуляризацию, то это означает, что регуляризация слишком сильная и мешает модели правильно аппроксимировать отдельные задачи. В таких случаях лучше увеличить емкость модели (т. е. размерность общих компонент сети). Если сеть MTL с k задачами нуждается в k-кратном увеличении емкости (или близко к тому) для поддержки всех задач, значит, предсказательная структура, вероятно, вообще не разделяется между задачами, и от идеи MTL лучше отказаться.

Если задачи очень тесно связаны, как, например, частеречная разметка и разбиение на блоки, то выигрыш от применения MTL может быть очень мал. Это особенно относится к случаю, когда сети обучаются на одном наборе данных, в котором каждое предложение аннотировано метками части речи и блока. Сеть разбиения может обучиться необходимому ей представлению и без помощи со стороны промежуточного частеречного учителя. Выигрыш от MTL начинает проявляться, когда обучающие данные для частеречной разметки и разбиения *не пересекаются* (но пользуются значительными по размеру общими частями словаря) или когда данные для частеречной разметки являются *надмножеством* данных для разбиения. В этой ситуации MTL позволяет дополнить сигнал учителя для задачи разбиения за счет обучения на данных с родственными метками для зада-

чи разметки. Поэтому часть сети, связанная с разбиением, сможет задействовать общее представление (и повлиять на него), обученное на метках частей речи в аннотациях к дополнительным данным.

20.3. Обучение с частичным привлечением учителя

С многозадачным и каскадным обучениями связано также *обучение с частичным привлечением учителя* (semi-supervised), когда имеется немного обучающих данных для интересующей нас задачи и дополнительные обучающие данные для других задач. Другие задачи могут обучаться с учителем или без него (например, сигналы учителя можно сгенерировать из неаннотированного корпуса, как в языковом моделировании, погружении слов или кодировании предложений (*см*. раздел 9.6, главу 10 и раздел 17.3)).

Мы хотели бы воспользоваться учителем для дополнительных задач (или придумать подходящие дополнительные задачи), чтобы улучшить верность предсказаний в главной задаче. Этот весьма распространенный сценарий – активная и важная область исследований: нам всегда не хватает аннотированных данных для интересных задач.

Обзор методов обучения обычных (не нейронных) сетей с частичным привлечением учителя в NLP *см*. в книге Søgaard [2013], опубликованной в этой серии.

В глубоких сетях обучение с частичным привлечением учителя, как и многозадачное обучение, можно выполнить, обучив на дополнительных задачах представление, которое затем используется в главной задаче в качестве добавочных входных данных или для инициализации. Например, можно предварительно обучить погружения слов или представления предложений на неаннотированных данных и воспользоваться ими для подачи на вход или инициализации частеречного разметчика, синтаксического анализатора или системы реферирования документов.

В некотором смысле мы занимались обучением с частичным привлечением учителя с того самого момента, как ввели дистрибутивные представления в виде предобученных погружений слов в главе 10. Иногда проблема располагает к более специализированному решению, как мы увидим в разделе 20.4.3. Сходство и связь с многозадачным обучением тоже понятны: мы используем аннотированные данные для одной задачи, чтобы улучшить качество предсказания в другой. Основное различие заключается в том, как различные задачи интегрируются в окончательную модель, а также в источнике аннотированных данных для разных задач, но граница между обоими подходами размыта. Вообще говоря, наверное, лучше не спорить о том, где проходят границы между каскадным обучением, многозадачным обучением и обучением с частичным привлечением учителя, а рассматривать их как взаимодополняющие и отчасти перекрывающиеся методы.

Из других подходов к обучению с частичным привлечением учителя отметим различные режимы, в которых одна или несколько моделей обучаются на небольшом аннотированном наборе данных, а затем используются для назначения меток большому набору неаннотированных данных. Потом автоматически аннотированные данные (возможно, после этапа контроля качества, основанного на согласии между моделями или других метрик доверия) используются для обучения

новой модели или выделения дополнительных признаков для существующей. Подобные подходы получили общее название – *самообучение*. В других методах задаются ограничения на решение, которое, по идее, должно направить модель в нужную сторону (например, указывается, что некоторые слова могут быть помечены только определенными метками или что каждое предложение должно содержать по меньшей мере одно слово с меткой X). Такие методы (пока) не адаптированы к нейронным сетям и выходят за рамки книги. Их обзор *см*. в книге Søgaard [2013].

20.4. Примеры

Теперь опишем несколько примеров, в которых эффективность MTL убедительно продемонстрирована.

20.4.1. Предсказание взгляда и сжатие предложений

В задаче сжатия предложения посредством удаления дано предложение, например «*Alan Turing, known as the father of computer science, the codebreaker that helped win World War 2, and the man tortured by the state for being gay, is to receive a pardon nearly 60 years after his death*»[1], и требуется породить его более короткий («сжатый»), но сохранивший основную информацию вариант путем удаления некоторых слов. Примером сжатия может служить «*Alan Turing is to receive a pardon*». Эту задачу можно смоделировать как глубокую biRNN, за которой следует МСП. При этом входами biRNN являются слова предложения, а выходами МСП – решения Keep (оставить) или Delete (удалить) для каждого слова.

В работе Klerke et al. [2016] мы показали, что качество задачи сжатия предложения посредством удаления можно повысить, воспользовавшись двумя дополнительными задачами предсказания последовательности: суперразметка с комбинаторными категориальными грамматиками (CCG) и предсказание взгляда. Обе задачи добавлены в архитектуру избирательного обобществления в виде отдельных МСП, которые получают данные от нижнего уровня biRNN.

В задаче CCG-суперразметки каждому слову назначается *CCG-суперметка* – сложная синтаксическая метка вида (S[dcl]nNP)/PP, отражающая его синтаксическую роль относительно остального предложения[2].

Задача предсказания взгляда – это когнитивная задача, связанная с тем, как люди читают написанный текст. Во время чтения взгляд перемещается по странице, задерживаясь на одних словах, пропуская другие и часто возвращаясь к уже прочитанным словам. Существует разделяемое многими мнение, что движение глаз во время чтения отражает механизмы обработки предложения мозгом, что, в свою очередь, отражает структуру предложения. Датчики движения глаз – это устройства, которые точно отслеживают движение глаз во время чтения. Доступны также корпусы текстов, в которых предложения сопровождаются точными из-

---

[1] Алан Тьюринг, известный как отец информатики, криптоаналитик, который помог выиграть Вторую мировую войну, и человек, преследуемый государством за гомосексуальность, должен получить помилование спустя почти 60 лет после смерти. – *Прим. перев.*

[2] Комбинаторные категориальные грамматики и CCG-суперметки выходят за рамки книги. Хорошим введением в изучение CCG в NLP является докторская диссертация Джулии Хокенмайер [Hockenmaier, 2003]. Понятие суперразметки введено в работе Joshi and Srinivas [1994].

мерениями движений глаз нескольких человек. В задаче предсказания взгляда сеть обучается предсказывать некоторые аспекты перемещения взгляда по тексту (как долго взгляд задержится на каждом слове, к каким словам взгляд будет возвращаться). Интуитивно кажется, что менее важные части предложения с большей вероятностью будут пропускаться или просматриваться мельком, а на более важных взгляд будет задерживаться дольше.

Данные для задач сжатия, синтаксической CCG-разметки и перемещения глаз никак не пересекаются, но мы обнаружили заметное улучшение верности сжатия после включения аннотированных данных для дополнительных задач.

20.4.2. Пометка дуг и синтаксический разбор

В этой книге мы не раз возвращались к архитектуре анализа зависимостей методом разложения на дуги. В частности, в разделе 16.2.3 были описаны признаки на основе biRNN, а в разделе 19.4.1 – система обучения структурным предсказаниям. Тогда мы описали *непомечающий анализатор* – модель, которая назначает оценку каждой возможной паре заглавное слово – модификатор, а на выходе возвращает набор дуг, представляющий наилучшее дерево разбора предложения. Однако функция оценки (и результирующие дуги) принимала во внимание только то, *какие* слова синтаксически связаны друг с другом, но не *природу связи* между словами.

Напомним (*см.* раздел 6.2.2), что дерево разбора зависимостей обычно содержит также информацию об отношении, выраженную в терминах метки зависимости на каждой дуге, например *det*, *prep*, *pobj*, *nsubj* и т. д. на рис. 20.6.

С помощью непомечающего анализатора метки дугам можно назначить, используя архитектуру, в которой biRNN читает слова предложения, а затем для дуги дерева (h, m) кодировки, найденные biRNN, конкатенируются и подаются на вход МСП для предсказания метки дуги.

Вместо того чтобы обучать отдельную сеть для предсказания меток, мы можем рассматривать *непомечающий анализ* и *пометку дуг* как родственные задачи в системе многозадачного обучения. Затем мы создаем одну biRNN, играющую роли пометчика дуг и анализатора, и используем закодированные состояния biRNN как входы для оценщика и пометчика дуг. На этапе обучения пометчик дуг будет видеть только золотые дуги (потому что для других, гипотетических дуг у нас нет информации о метках), а оценщик дуг – все возможные дуги.

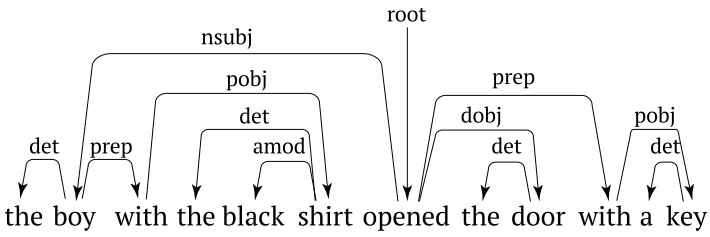


Рис. 20.6 ❖ Помеченное дерево зависимостей

И действительно, в работе Kiperwasser and Goldberg [2016b] показано, что эти задачи тесно связаны. Обучение совместной сети для выполнения как непомечающей оценки дуг, так и пометки дуг, в которой используется общий biRNN-

кодировщик, не только повышает верность пометки дуг, но и *значительно улучшает верность непомеченных деревьев разбора.*

20.4.3. Разрешение лексической многозначности предлогов и предсказание перевода предлогов

Рассмотрим задачу о разрешении лексической многозначности предлогов из раздела 7.6. Напомним, что это проблема слова в контексте, в которой требуется назначить каждому предлогу одну из K возможных *меток смысла*: Manner (образ действия), Purpose (цель), Location (местоположение), Duration (продолжительность) и т. д.). Существуют аннотированные корпусы для этой задачи [Litkowski and Hargraves, 2007, Schneider et al., 2016], но они небольшие.

В разделе 7.6 мы обсуждали богатый набор базовых признаков, которые можно использовать для обучения модели разрешения многозначности предлогов. Обозначим функцию выделения признаков, которая принимает предлог и возвращает кодировку этих признаков в виде вектора $\phi_{\text{sup}}(s, i)$, где s – входное предложение (включая слова, метки частей речи, леммы и дерево синтаксического разбора), а i – индекс предлога в предложении. Функция ϕ_{sup}, основанная на признаках типа тех, что мы видели в разделе 7.6, без признаков, взятых из WordNet, но с предобученными погружениями слов, – сильный экстрактор. Подача ее результатов на вход МСП для предсказания работает сравнительно неплохо (хотя верность все же удручающе мала, меньше 80 %), а попытки заменить или дополнить ее экстрактором на основе biRNN не приводят к повышению верности.

Здесь мы покажем, как можно улучшить качество предсказания смысла предлогов, воспользовавшись подходом *с частичным привлечением учителя*, основанным на обучении полезного представления на больших объемах неаннотированных данных, которое мы превратим в набор взаимосвязанных и полезных задач предсказания.

Точнее, мы будем использовать задачи, выведенные из *многоязычных данных, выровненных по предложениям*. Это будут пары, состоящие из английского предложения и его перевода на другой язык[1]. При переводе с английского предлогу могут соответствовать различные варианты. Выбор предлога на другом языке зависит от смысла английского предлога, отраженного в контексте объемлющего предложения. Хотя предлоги на всех языках неоднозначны, характер этой неоднозначности варьируется. Поэтому предсказание того, каким иностранным предлогом следует перевести английский, основанное на сентенциональном контексте, – хорошая вспомогательная задача для разрешения неоднозначности предлогов. Такой подход принят в работе Gonen and Goldberg [2016]. Мы дадим лишь высокоуровневый обзор, детали смотрите в оригинальной статье.

Обучающие данные созданы на основе многоязычного параллельного корпуса выровненных предложений. Корпус выровнен по словам с применением алгоритма, описанного в работе [Dyer et al., 2013], и в качестве обучающих примеров выделены кортежи ⟨предложение, позиция предлога, иностранный язык, иностранный предлог⟩. Получив кортеж $s = \langle w_{1:n}, i, L, f \rangle$, задача должна предсказать перевод

[1] Такие ресурсы легко получить, например протоколы заседаний органов Европейского союза (корпус Europarl, [Koehn, 2005]), или найти в вебе [Uszkoreit et al., 2010]. Именно подобные ресурсы лежат в основе статистического машинного перевода.

предлога w_i в контексте предложения s. Возможные выходы берутся из множества зависящих от языка вариантов p_L, а правильным выходом является f.

Хочется надеяться, что представление контекста w_i, подходящее для предсказания иностранного предлога f, будет полезно и для предсказания смысла предлога. Мы моделируем задачу как *кодировщик* Enc(s, i), который кодирует сентенциональный контекст w_i вектором, и предиктор, пытающийся предсказать правильный предлог. Кодировщик очень похож на biRNN, но не включает сам предлог, чтобы заставить сеть обращать больше внимания на контекст, а предиктор – это специфичный для конкретного языка МСП:

$$\begin{aligned} p(\text{foreign} = f | s, i, L) &= \text{softmax}(\text{MLP}^L_{\text{foreign}}(\text{Enc}(s, i)))_{[f]}; \\ \text{Enc}(s, i) &= [\text{RNN}^f(w_{1:i-1}); \text{RNN}^b(w_{n:i+1})]. \end{aligned} \tag{20.8}$$

Кодировщик является общим для разных языков. После обучения сети на нескольких миллионах пар ⟨английское предложение, иностранный предлог⟩ мы получили предобученный кодировщик контекста, который можно впоследствии использовать в сети для разрешения лексической многозначности предлогов, конкатенировав его с аннотированным представлением признаков. Тогда наш разрешитель многозначности с частичным привлечением учителя принимает вид:

$$p(\text{sense} = j | s, i) = \text{softmax}(\text{MLP}_{\text{sup}}([\phi_{\text{sup}}(s, i); \text{Enc}(s, i)))_{[j]}, \tag{20.9}$$

где Enc – предобученный кодировщик, который дополнительно обучен сетью предсказания смысла, а ϕ_{sup} – экстрактор признаков, обученный с учителем. Этот подход систематически повышает верность предсказания смысла на 1–2 процентных пункта в зависимости от конфигурационных параметров[1].

20.4.4. Условная генерация: многоязычный машинный перевод, синтаксический анализ и описание изображений

Многозадачное обучение легко встраивается в каркас кодировщик–декодер. В работе Luong et al. [2016] это продемонстрировано в контексте машинного перевода. Описанная там система перевода построена на основе архитектуры последовательность-в-последовательность (раздел 17.2.1) без механизма внимания. Существуют системы перевода и получше (и прежде всего те, в которых механизм внимания используется), но целью работы было показать, что многозадачное обучение действительно способно привести к улучшению.

Луонг с коллегами исследуют различные конфигурации многозадачного обучения в своей системе. В первой конфигурации (один ко многим) кодировщик (англоязычных предложений в векторы) разделяемый и используется с двумя разными декодерами: один генерирует перевод на немецкий, а другой создает линеаризованные деревья разбора английских предложений (т. е. для предложения *the boy opened the door* должна быть предсказана последовательность (S (NP DT NN) (VP

[1] Хотя повышение верности на 1–2 процента не выглядит очень уж впечатляюще, это, к сожалению, максимум того, на что можно надеяться в современных сценариях с частичным привлечением учителя, когда эталонная система, обученная с учителем, уже достаточно сильна. Улучшения по сравнению с более слабыми эталонными системами значительнее.

VBD (NP DT NN))))). Система обучалась на параллельном корпусе пар ⟨английский, немецкий⟩ и на золотых деревьях разбора из базы данных Penn Treebank [Marcus et al., 1993]. Данные для перевода и синтаксического анализа не пересекаются. Благодаря многозадачной постановке общий кодировщик обучается порождать векторы, информативные для обеих задач. Многозадачная сеть кодировщик–декодер эффективна: сеть, обученная для двух задач (один кодировщик, два декодера), работает лучше, чем отдельные сети, содержащие только одну пару кодировщик–декодер. Предположительно эта система работает, потому что кодирование базовых элементов синтаксической структуры предложения дает информацию для выбора порядка слов и синтаксической структуры перевода, и наоборот. Задачи перевода и синтаксического анализа действительно обладают синергией.

Во второй конфигурации (многие к одному) имеется единственный декодер, но несколько разных кодировщиков. В качестве задач выбраны машинный перевод (с немецкого на английский) и описание изображений (генерация подписей на английском языке). Задача декодера – порождать англоязычные предложения. Один кодировщик кодирует немецкие предложения, другой – изображения. Как и раньше, наборы данных для перевода и описания изображений не пересекаются. И снова после настройки параметров совместное обучение системы оказывается лучше обучения систем по отдельности, хотя выигрыш несколько меньше. В данном случае нет очевидной связи между задачей кодирования немецких предложений (для которых характерны запутанные предикации и сложная синтаксическая структура) и задачей кодирования содержания изображений (где требуется закодировать основные компоненты простых сцен). Выигрыш, видимо, проистекает из того факта, что обе задачи играют роль учителя для той части сети-декодера, которая связана с языковым моделированием, что позволяет порождать лучше сформированные англоязычные предложения. Кроме того, улучшение может объясняться эффектом регуляризации, благодаря которому одна пара ⟨кодировщик, декодер⟩ препятствует переобучению другой пары на своих обучающих данных.

Несмотря на относительно низкий базовый уровень, результаты, полученные в работе Luong et al. [2016], вселяют надежду, поскольку наводят на мысль, что многозадачное обучение в системах условной генерации все-таки может давать выигрыш при правильном подборе синергетических задач.

20.5. Перспективы

Каскадное обучение, многозадачное обучение и обучение с частичным привлечением учителя – перспективные методы. Нейронные сети, в основе которых лежит градиентное обучение на графе вычислений, открывают немало возможностей их органичного использования. Во многих случаях такие подходы дают реальное и систематически подтверждаемое повышение верности предсказаний. К сожалению, на момент написания этой книги выигрыш часто оказывается относительно скромным по сравнению с качеством эталонной системы, особенно если эталонная планка поднята высоко. Но это не должно стать причиной для отказа от этих методов, поскольку зачастую выигрыш все-таки вполне ощутим. Это должно подтолкнуть к активной работе по их совершенствованию и уточнению, так что в будущем мы ожидаем более убедительных результатов.

Глава 21

Заключение

21.1. Что мы узнали?

Введение нейросетевых методов преобразило NLP. Оно дало импульс к переходу от линейных моделей с ручным конструированием признаков (в том числе способов отката и комбинационных признаков) к многослойным перцептронам, которые сами обучаются комбинациям признаков (как обсуждалось в первой части книги); к архитектурам типа сверточных нейронных сетей, которые способы выявить допускающие обобщение *n*-граммы и *n*-граммы с пропусками (*см*. главу 13); к архитектурам типа РНС и двунаправленных РНС (главы 14–16), которые могут выявить тонкие паттерны и закономерности в последовательностях произвольной длины; и к рекурсивным нейронным сетям (глава 18), умеющим представлять деревья. Они также принесли с собой методы кодирования слов векторами, основанные на дистрибутивном сходстве, которые могут быть эффективны для обучения с частичным привлечением учителя (главы 10–11), а также методы немарковского языкового моделирования, которые, в свою очередь, проложили путь к гибким лингвистическим моделям условной генерации (глава 17) и произвели революцию в машинном переводе. Нейросетевые методы предлагают также много возможностей для многозадачного обучения (глава 20). Более того, традиционные, возникшие до появления нейронных сетей методы структурного предсказания легко адаптируются к включению экстракторов признаков и предикторов, основанных на нейронных сетях (глава 19).

21.2. Что ждет впереди?

В общем и целом в этой области наблюдается очень быстрый прогресс, и трудно предсказать, что нам грядущее готовит. Но одно ясно, по крайней мере на мой взгляд, – при всех своих впечатляющих достижениях нейронные сети не являются панацеей для понимания и порождения естественного языка. Хотя они во многом улучшают статистические методы NLP прошлого поколения, главные проблемы остаются: язык по-прежнему дискретный и неоднозначный, мы толком не понимаем, как он работает, и маловероятно, что нейронная сеть сможет обучиться всем тонкостям самостоятельно без руководства со стороны человека. Проблемы, упомянутые во введении, никуда не денутся и после внедрения нейросетевых методов, а знакомство с лингвистическими идеями и ресурсами, описанными в главе 6, так же важно, как и раньше, если мы хотим проектировать хорошие системы

обработки языка. Качество решения многих задач NLP, даже таких низкоуровневых и, на первый взгляд, простых, как разрешение кореференции местоимений [Clark and Manning, 2016, Wiseman et al., 2016] или разрешение неоднозначности границ координации [Ficler and Goldberg, 2016], еще очень далеко от идеального. Проектирование систем обучения, нацеленных на такие низкоуровневые задачи понимания языка, остается столь же важной исследовательской задачей, как и до появления нейросетевых методов NLP.

Еще одна важная проблема – непрозрачность обученных представлений и отсутствие строгой теории, стоящей за архитектурами и алгоритмами обучения. Для движения вперед остро необходимы исследования по интерпретируемости нейросетевых представлений, также необходимо лучше понимать емкость обучения и динамику обучения различных архитектур.

На момент написания этой книги нейронные сети, по существу, остаются методами обучения с учителем и нуждаются в относительно больших объемах аннотированных обучающих данных. Хотя применение предобученных погружений слов создает удобную платформу для обучения с частичным привлечением учителя, мы все еще очень плохо умеем использовать непомеченные данные с целью уменьшить зависимость от наличия аннотированных примеров. Напомним, что человеку зачастую для обобщения нужна лишь горстка примеров, тогда как нейронным сетям для приличной работы обычно необходимы по меньшей мере сотни помеченных примеров, даже если речь идет о самых простых лингвистических задачах. Отыскание способов эффективно задействовать скромные объемы аннотированных данных вкупе с большими объемами неаннотированных, а также выполнять обобщение между разными предметными областями, скорее всего, произведет еще одну революцию в отрасли.

Наконец, упомянем один вопрос, который лишь походя затрагивался в этой книге, – язык не является изолированным явлением. Когда человек изучает, воспринимает и порождает языковые конструкции, он апеллирует к реальному миру, а высказывания, сформулированные на естественном языке, чаще всего связаны с реальными сущностями или практическим опытом. Изучение языка в конкретном контексте, либо подкрепленное другими модальностями, например изображениями, видео или управлением роботом, либо как часть агента, взаимодействующего с миром для достижения определенных целей, – еще одна многообещающая тема, находящаяся на самом переднем крае науки.

Список литературы

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. http://tensorflow.org/.

Heike Adel, Ngoc Thang Vu and Tanja Schultz. Combination of recurrent neural networks and factored language models for code-switching language modeling. In Proc. of the 51st Annual Meeting of the Association for Computational Linguistics (volume 2: Short Papers), p. 206–211, Sofia, Bulgaria, August 2013.

Roee Aharoni, Yoav Goldberg and Yonatan Belinkov. Proc. of the 14th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology, chapter improving sequence to sequence learning for morphological inflection generation: The BIU-MIT systems for the SIGMORPHON 2016 shared task for morphological reinflection, p. 41–48. Association for Computational Linguistics, 2016. http://aclweb.org/anthology/W16-2007 DOI:10.18653/v1/W16-2007.

Roee Aharoni and Yoav Goldberg. Towards string-to-tree neural machine translation. Proc. Of ACL, 2017.

M. A. Aizerman, E. A. Braverman and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. In Automation and Remote Control, number 25 in Automation and Remote Control, p. 821–837, 1964.

Erin L. Allwein, Robert E. Schapire and Yoram Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. Journal of Machine Learning Research, 1:113–141, 2000.

Rie Ando and Tong Zhang. A high-performance semi-supervised learning method for text chunking. In Proc. of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05), p. 1–9, Ann Arbor, Michigan, June 2005a. DOI: 10.3115/1219840.1219841.

Rie Kubota Ando and Tong Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. The Journal of Machine Learning Research, 6:1817–1853, 2005b.

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov and Michael Collins. Globally normalized transition-based neural networks. In Proc. of the 54th Annual Meeting of the Association for Computational Linguistics (volume 1: Long Papers), p. 2442–2452, 2016. http://aclweb.org/anthology/P16-1231 DOI: 10.18653/v1/P16-1231.

Michael Auli and Jianfeng Gao. Decoder integration and expected BLEU training for recurrent neural network language models. In Proc. of the 52nd Annual Meeting of the Association for Computational Linguistics (volume 2: Short Papers), p. 136–142, Baltimore, Maryland, June 2014. DOI: 10.3115/v1/p14-2023.

Michael Auli, Michel Galley, Chris Quirk and Geoffrey Zweig. Joint language and translation modeling with recurrent neural networks. In Proc. of the 2013 Conference on Empirical Methods in Natural Language Processing, p. 1044–1054, Seattle, Washington. Association for Computational Linguistics, October 2013.

Oded Avraham and Yoav Goldberg. The interplay of semantics and morphology in word embeddings. *EACL*, 2017.

Dzmitry Bahdanau, Kyunghyun Cho and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. arXiv:1409.0473 [cs, stat], September 2014.

Miguel Ballesteros, Chris Dyer and Noah A. Smith. Improved transition-based parsing by modeling characters instead of words with LSTMs. In Proc. of the 2015 Conference on Empirical Methods in Natural Language Processing, p. 349–359, Lisbon, Portugal. Association for Computational Linguistics, September 2015. DOI: 10.18653/v1/d15-1041.

Miguel Ballesteros, Yoav Goldberg, Chris Dyer and Noah A. Smith. Training with exploration improves a greedy stack-LSTM parser, EMNLP 2016. arXiv:1603.03793 [cs], March 2016. DOI: 10.18653/v1/d16-1211.

Mohit Bansal, Kevin Gimpel and Karen Livescu. Tailoring continuous word representations for dependency parsing. In Proc. of the 52nd Annual Meeting of the Association for Computational Linguistics (volume 2: Short Papers), p. 809–815, Baltimore, Maryland, June 2014. DOI: 10.3115/v1/p14-2131.

Marco Baroni and Alessandro Lenci. Distributional memory: A general framework for corpus-based semantics. Computational Linguistics, 36(4):673–721, 2010. DOI: 10.1162/coli_a_00016.

Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul and Jeffrey Mark Siskind. Automatic differentiation in machine learning: A survey. arXiv:1502.05767 [cs], February 2015.

Emily M. Bender. Linguistic Fundamentals for Natural Language Processing: 100 Essentials from Morphology and Syntax. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2013.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. *CoRR*, abs/1506.03099, 2015. http: //arxiv.org/abs/1506.03099.

Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. arXiv:1206.5533 [cs], June 2012. DOI: 10.1007/978-3-642-35289-8_26.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent and Christian Janvin. A neural probabilistic language model. Journal of Machine Learning Research, 3:1137–1155, March 2003. ISSN 1532-4435. DOI: 10.1007/10985687_6.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert and Jason Weston. Curriculum learning. In Proc. of the 26th Annual International Conference on Machine Learning, p. 41–48. ACM, 2009. DOI: 10.1145/1553374.1553380.

Yoshua Bengio, Ian J. Goodfellow and Aaron Courville. Deep Learning. MIT Press, 20161.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In Proc. of the Python for Scientific Computing Conference (SciPy), June 2010.

Jeff A. Bilmes and Katrin Kirchhoff. Factored language models and generalized parallel backoff. In Companion Volume of the Proc. of HLT-NAACL–Short Papers, 2003. DOI: 10.3115/1073483.1073485.

Zsolt Bitvai and Trevor Cohn. Non-linear text regression with a deep convolutional neural network. In Proc. of the 53rd Annual Meeting of the Association for Computa-

[1] *Иошуа Бенджио, Ян Гудфеллоу, Аарон Курвилль.* Глубокое обучение. М.: ДМК Пресс, 2017.

tional Linguistics and the 7th International Joint Conference on Natural Language Processing (volume 2: Short Papers), p. 180–185, Beijing, China, July 2015. DOI: 10.3115/v1/p15-2030.

Tolga Bolukbasi, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama and Adam Tauman Kalai. Quantifying and reducing stereotypes in word embeddings. CoRR, abs/1606.06121, 2016. http://arxiv.org/abs/1606.06121.

Bernhard E. Boser, Isabelle M. Guyon and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In Proc. of the 5th Annual ACM Workshop on Computational Learning Theory, p. 144–152. ACM Press, 1992. DOI: 10.1145/130385.130401.

Jan A. Botha and Phil Blunsom. Compositional morphology for word representations and language modelling. In Proc. of the 31st International Conference on Machine Learning (ICML), Beijing, China, June 2014.

Léon Bottou. Stochastic gradient descent tricks. In Neural Networks: Tricks of the Trade, p. 421–436. Springer, 2012. DOI: 10.1007/978-3-642-35289-8_25.

R. Samuel Bowman, Gabor Angeli, Christopher Potts and D. Christopher Manning. A large annotated corpus for learning natural language inference. In Proc. of the 2015 Conference on Empirical Methods in Natural Language Processing, p. 632–642. Association for Computational Linguistics, 2015. http://aclweb.org/anthology/D15-1075 DOI: 10.18653/v1/D15-1075.

Peter Brown, Peter deSouza, Robert Mercer, T. Watson, Vincent Della Pietra and Jenifer Lai. Class-based n-gram models of natural language. Computational Linguistics, 18(4), December 1992. http://aclweb.org/anthology/J92-4003.

John A. Bullinaria and Joseph P. Levy. Extracting semantic representations from word cooccurrence statistics: A computational study. Behavior Research Methods, 39(3):510–526, 2007. DOI: 10.3758/bf03193020.

A. Caliskan-Islam, J. J. Bryson and A. Narayanan. Semantics derived automatically from language corpora necessarily contain human biases. CoRR, abs/1608.07187, 2016.

Rich Caruana. Multitask learning. Machine Learning, 28:41–75, 1997. DOI: 10.1007/978-1-4615-5529-2_5.

Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In Proc. of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05), p. 173–180, Ann Arbor, Michigan, June 2005. DOI: 10.3115/1219840.1219862.

Danqi Chen and Christopher Manning. A fast and accurate dependency parser using neural networks. In Proc. of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), p. 740–750, Doha, Qatar. Association for Computational Linguistics, October 2014. DOI: 10.3115/v1/d14-1082.

Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. In 34th Annual Meeting of the Association for Computational Linguistics, 1996. http: //aclweb.org/anthology/P96-1041 DOI: 10.1006/csla.1999.0128.

Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. Computer Speech and Language, 13(4):359–394, 1999. DOI: 10.1006/csla.1999.0128.

Wenlin Chen, David Grangier and Michael Auli. Strategies for training large vocabulary neural language models. In Proc. of the 54th Annual Meeting of the Association for Computational Linguistics (volume 1: Long Papers), p. 1975–1985, 2016. http://aclweb.org/anthology/P16-1186 DOI: 10.18653/v1/P16-1186.

Yubo Chen, Liheng Xu, Kang Liu, Daojian Zeng and Jun Zhao. Event extraction via dynamic multi-pooling convolutional neural networks. In Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (volume 1: Long Papers), p. 167–176, Beijing, China, July 2015. DOI: 10.3115/v1/p15-1017.

Kyunghyun Cho. Natural language understanding with distributed representation. arXiv:1511.07916 [cs, stat], November 2015.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In Proc. of SSST-8, 8th Workshop on Syntax, Semantics and Structure in Statistical Translation, p. 103–111, Doha, Qatar. Association for Computational Linguistics, October 2014a. DOI: 10.3115/v1/w14-4012.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Proc. of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), p. 1724–1734, Doha, Qatar. Association for Computational Linguistics, October 2014b. DOI: 10.3115/v1/d14-1179.

Do Kook Choe and Eugene Charniak. Parsing as language modeling. In Proc. of the Conference on Empirical Methods in Natural Language Processing, p. 2331–2336, Austin, Texas. Association for Computational Linguistics, November 2016. https://aclweb.org/anthology/D16-1257 DOI: 10.18653/v1/d16-1257.

Grzegorz Chrupala. Normalizing tweets with edit scripts and recurrent neural embeddings. In Proc. of the 52nd Annual Meeting of the Association for Computational Linguistics (volume 2: Short Papers), p. 680–686, Baltimore, Maryland, June 2014. DOI: 10.3115/v1/p14-2111.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv:1412.3555 [cs], December 2014.

Junyoung Chung, Kyunghyun Cho and Yoshua Bengio. A character-level decoder without explicit segmentation for neural machine translation. In Proc. of the 54th Annual Meeting of the Association for Computational Linguistics (volume 1: Long Papers), p. 1693–1703, 2016. http://aclweb.org/anthology/P16-1160 DOI: 10.18653/v1/P16-1160.

Kenneth Ward Church and Patrick Hanks. Word association norms, mutual information, and lexicography. Computational Linguistics, 16(1):22–29, 1990. DOI: 10.3115/981623.981633.

Kevin Clark and Christopher D. Manning. Improving coreference resolution by learning entitylevel distributed representations. In Association for Computational Linguistics (ACL), 2016. /u/apache/htdocs/static/pubs/clark2016improving.pdf DOI: 10.18653/v1/p16-1061.

Michael Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In Proc. of the Conference on Empirical Methods in Natural Language Processing, p. 1–8. Association for Computational Linguistics, July 2002. DOI: 10.3115/1118693.1118694.

Michael Collins and Terry Koo. Discriminative reranking for natural language parsing. Computational Linguistics, 31(1):25–70, March 2005. ISSN 0891-2017. DOI: 10.1162/0891201053630273.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In Proc. of the 25th International Conference on Machine Learning, p. 160–167. ACM, 2008. DOI: 10.1145/1390156.1390177.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu and Pavel Kuksa. Natural language processing (almost) from scratch. Pe Journal of Machine Learning Research, 12:2493–2537, 2011.

Alexis Conneau, Holger Schwenk, Loïc Barrault and Yann LeCun. Very deep convolutional networks for natural language processing. CoRR, abs/1606.01781, 2016. http://arxiv.org/abs/1606.01781.

Ryan Cotterell and Hinrich Schutze. Morphological word embeddings. *NAACL*, 2015.

Ryan Cotterell, Christo Kirov, John Sylak-Glassman, David Yarowsky, Jason Eisner and Mans Hulden. Proc. of the 14th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology, chapter The SIGMORPHON 2016 Shared Task–Morphological Reinflection, p. 10–22. Association for Computational Linguistics, 2016. http://aclweb.org/anthology/W16-2002 DOI: 10.18653/v1/W16-2002.

Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernelbased vector machines. The Journal of Machine Learning Research, 2:265–292, 2002.

Mathias Creutz and Krista Lagus. Unsupervised models for morpheme segmentation and morphology learning. ACM Transactions of Speech and Language Processing, 4(1):3:1–3:34, February 2007. ISSN 1550-4875. DOI: 10.1145/1187415.1187418.

James Cross and Liang Huang. Incremental parsing with minimal features using bidirectional LSTM. In Proc. of the 54th Annual Meeting of the Association for Computational Linguistics (volume 2: Short Papers), p. 32–37, 2016a. http://aclweb.org/anthology/P16-2006 DOI: 10.18653/v1/P16-2006.

James Cross and Liang Huang. Span-based constituency parsing with a structure-label system and dynamic oracles. In Proc. of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics, 2016b. DOI: 10.18653/v1/d16-1001.

G. Cybenko. Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems, 2(4):303–314, December 1989. ISSN 0932-4194, 1435-568X. DOI: 10.1007/BF02551274.

Ido Dagan and Oren Glickman. Probabilistic textual entailment: Generic applied modeling of language variability. In PASCAL Workshop on Learning Methods for Text Understanding and Mining, 2004.

Ido Dagan, Fernando Pereira and Lillian Lee. Similarity-based estimation of word cooccurrence probabilities. In ACL, 1994. DOI: 10.3115/981732.981770.

Ido Dagan, Oren Glickman and Bernardo Magnini. The PASCAL recognising textual entailment challenge. In Machine Learning Challenges, Evaluating Predictive Uncertainty, Visual Object Classification and Recognizing Textual Entailment, First PASCAL Machine Learning Challenges Workshop, MLCW, p. 177–190, Southampton, UK, April 11–13, 2005. (revised selected papers). DOI: 10.1007/11736790_9.

Ido Dagan, Dan Roth, Mark Sammons and Fabio Massimo Zanzotto. Recognizing Textual Entailment: Models and Applications. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2013. DOI: 10.2200/s00509ed1v01y201305hlt023.

G. E. Dahl, T. N. Sainath and G. E. Hinton. Improving deep neural networks for LVCSR using rectified linear units and dropout. In 2013 IEEE International Conference

on Acoustics, Speech and Signal Processing (ICASSP), p. 8609–8613, May 2013. DOI: 10.1109/ICASSP.2013.6639346.

Hal Daumé III, John Langford and Daniel Marcu. Search-based structured prediction. Machine Learning Journal (MLJ), 2009. DOI: 10.1007/s10994-009-5106-x.

Hal Daumé III. A Course In Machine Learning. Self Published, 2015.

Yann N. Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional nonconvex optimization. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., Advances in Neural Information Processing Systems 27, p. 2933–2941. Curran Associates, Inc., 2014.

Adrià de Gispert, Gonzalo Iglesias and Bill Byrne. Fast and accurate preordering for SMT using neural networks. In Proc. of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, p. 1012–1017, Denver, Colorado, 2015. DOI: 10.3115/v1/n15-1105.

Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz and John Makhoul. Fast and robust neural network joint models for statistical machine translation. In Proc. of the 52nd Annual Meeting of the Association for Computational Linguistics (volume 1: Long Papers), p. 1370–1380, Baltimore, Maryland, June 2014. DOI: 10.3115/v1/p14-1129.

Trinh Do, Thierry Arti and others. Neural conditional random fields. In International Conference on Artificial Intelligence and Statistics, p. 177–184, 2010.

Pedro Domingos. The Master Algorithm. Basic Books, 2015.

Li Dong, Furu Wei, Chuanqi Tan, Duyu Tang, Ming Zhou and Ke Xu. Adaptive recursive neural network for target-dependent twitter sentiment classification. In Proc. of the 52nd Annual Meeting of the Association for Computational Linguistics (volume 2: Short Papers), p. 49–54, Baltimore, Maryland, June 2014. DOI: 10.3115/v1/p14-2009.

Li Dong, Furu Wei, Ming Zhou and Ke Xu. Question answering over freebase with multicolumn convolutional neural networks. In Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (volume 1: Long Papers), p. 260–269, Beijing, China, July 2015. DOI: 10.3115/v1/p15-1026.

Cicero dos Santos and Maira Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In Proc. of COLING, the 25th International Conference on Computational Linguistics: Technical Papers, p. 69–78, Dublin City University, Dublin, Ireland. Association for Computational Linguistics, August 2014.

Cicero dos Santos and Bianca Zadrozny. Learning character-level representations for part-of-speech tagging. In Proc. of the 31st International Conference on Machine Learning (ICML), p. 1818–1826, 2014.

Cicero dos Santos, Bing Xiang and Bowen Zhou. Classifying relations by ranking with convolutional neural networks. In Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (volume 1: Long Papers), p. 626–634, Beijing, China, July 2015. DOI: 10.3115/v1/p15-1061.

John Duchi, Elad Hazan and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. The Journal of Machine Learning Research, 12:2121–2159, 2011.

Kevin Duh, Graham Neubig, Katsuhito Sudoh and Hajime Tsukada. Adaptation data selection using neural language models: experiments in machine translation. In Proc.

of the 51st Annual Meeting of the Association for Computational Linguistics (volume 2: Short Papers), p. 678–683, Sofia, Bulgaria, August 2013.

Greg Durrett and Dan Klein. Neural CRF parsing. In Proc. of the 53rd Annual Meeting of theAssociation for Computational Linguistics and the 7th International Joint Conference on NaturalLanguage Processing (volume 1: Long Papers), p. 302–312, Beijing, China, July 2015. DOI: 10.3115/v1/p15-1030.

Chris Dyer, Victor Chahuneau and A. Noah Smith. A simple, fast, and effective reparameterization of IBM model 2. In Proc. of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, p. 644–648, 2013. http://aclweb.org/anthology/N13-1073.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews and Noah A. Smith. Transition-based dependency parsing with stack long short-term memory. In Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (volume 1: Long Papers), p. 334–343, Beijing, China, July 2015. DOI: 10.3115/v1/p15-1033.

C. Eckart and G. Young. The approximation of one matrix by another of lower rank. Psychometrika, 1:211–218, 1936. DOI: 10.1007/bf02288367.

Jason Eisner and Giorgio Satta. Efficient parsing for bilexical context-free grammars and head automaton grammars. In Proc. of the 37th Annual Meeting of the Association for Computational Linguistics, 1999. http://aclweb.org/anthology/P99-1059 DOI: 10.3115/1034678.1034748.

Jeffrey L. Elman. Finding structure in time. Cognitive Science, 14(2):179–211, March 1990. ISSN 1551-6709. DOI: 10.1207/s15516709cog1402_1.

Martin B. H. Everaert, Marinus A. C. Huybregts, Noam Chomsky, Robert C. Berwick and Johan J. Bolhuis. Structures, not strings: Linguistics as part of the cognitive sciences. Trends in Cognitive Sciences, 19(12):729–743, 2015. DOI: 10.1016/j.tics.2015.09.008.

Manaal Faruqui and Chris Dyer. Improving vector space word representations using multilingual correlation. In Proc. of the 14th Conference of the European Chapter of the Association for Computational Linguistics, p. 462–471, Gothenburg, Sweden, April 2014. DOI: 10.3115/v1/e14-1049.

Manaal Faruqui, Jesse Dodge, Kumar Sujay Jauhar, Chris Dyer, Eduard Hovy and A. Noah Smith. Retrofitting word vectors to semantic lexicons. In Proc. of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, p. 1606–1615, 2015. http://aclweb.org/anthology/N15-1184 DOI: 10.3115/v1/N15-1184.

Manaal Faruqui, Yulia Tsvetkov, Graham Neubig and Chris Dyer. Morphological inflection generation using character sequence to sequence learning. In Proc. of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, p. 634–643, 2016. http://aclweb.org/anthology/N16-1077 DOI: 10.18653/v1/N16-1077.

Christiane Fellbaum. WordNet: An Electronic Lexical Database. Bradford Books, 1998.

Jessica Ficler and Yoav Goldberg. A neural network for coordination boundary prediction. In Proc. of the 2016 Conference on Empirical Methods in Natural Language Processing, p. 23–32, Austin, Texas. Association for Computational Linguistics, November 2016. https://aclweb.org/anthology/D16-1003 DOI: 10.18653/v1/d16-1003.

Katja Filippova and Yasemin Altun. Overcoming the lack of parallel data in sentence compression. In Proc. of the 2013 Conference on Empirical Methods in Natural Lan-

guage Processing, p. 1481–1491. Association for Computational Linguistics, 2013. http://aclweb.org/anthology/D13-1155.

Katja Filippova, Enrique Alfonseca, Carlos A. Colmenares, Lukasz Kaiser and Oriol Vinyals. Sentence compression by deletion with LSTMs. In Proc. of the 2015 Conference on Empirical Methods in Natural Language Processing, p. 360–368, Lisbon, Portugal. Association for Computational Linguistics, September 2015. DOI: 10.18653/v1/d15-1042.

Charles J. Fillmore, Josef Ruppenhofer and Collin F. Baker. FrameNet and representing the link between semantic and syntactic relations. Language and Linguistics Monographs Series B, p. 19–62, Institute of Linguistics, Academia Sinica, Taipei, 2004.

John R. Firth. A synopsis of linguistic theory 1930–1955. In Studies in Linguistic Analysis, Special volume of the Philological Society, p. 1–32. Firth, John Rupert, Haas William, Halliday, Michael A. K., Oxford, Blackwell Ed., 1957.

John R. Firth. The technique of semantics. Transactions of the Philological Society, 34(1):36–73, 1935. ISSN 1467-968X. DOI: 10.1111/j.1467-968X.1935.tb01254.x.

Mikel L. Forcada and Ramón P. Ñeco. Recursive hetero-associative memories for translation. In Biological and Artificial Computation: From Neuroscience to Technology, p. 453–462. Springer, 1997. DOI: 10.1007/bfb0032504.

Philip Gage. A new algorithm for data compression. C Users Journal, 12(2):23–38, February 1994. ISSN 0898-9788. http://dl.acm.org/citation.cfm?id=177910.177914.

Yarin Gal. A theoretically grounded application of dropout in recurrent neural networks. *CoRR*, abs/1512.05287, December 2015.

Kuzman Ganchev and Mark Dredze. Proc. of the ACL-08: HLT Workshop on Mobile Language Processing, chapter Small Statistical Models by Random Feature Mixing, p. 19–20. Association for Computational Linguistics, 2008. http://aclweb.org/anthology/W08-0804.

Juri Ganitkevitch, Benjamin Van Durme and Chris Callison-Burch. PPDB: The paraphrase database. In Proc. of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, p. 758–764, 2013. http://aclweb.org/anthology/N13-1092.

Jianfeng Gao, Patrick Pantel, Michael Gamon, Xiaodong He and Li Deng. Modeling interestingness with deep neural networks. In Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP), p. 2–13, Doha, Qatar. Association for Computational Linguistics, October 2014. DOI: 10.3115/v1/d14-1002.

Dan Gillick, Cliff Brunk, Oriol Vinyals and Amarnag Subramanya. Multilingual language processing from bytes. In Proc. of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, p. 1296–1306, 2016. http://aclweb.org/anthology/N16-1155 DOI: 10.18653/v1/N16-1155.

Jesús Giménez and Lluis Màrquez. SVMTool: A general POS tagger generator based on support vector machines. In Proc. of the 4th LREC, Lisbon, Portugal, 2004.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In International Conference on Artificial Intelligence and Statistics, p. 249–256, 2010.

Xavier Glorot, Antoine Bordes and Yoshua Bengio. Deep sparse rectifier neural networks. In International Conference on Artificial Intelligence and Statistics, p. 315–323, 2011.

Yoav Goldberg. A primer on neural network models for natural language processing. Journal of Artificial Intelligence Research, 57:345–420, 2016.

Yoav Goldberg and Michael Elhadad. An efficient algorithm for easy-first non-directional dependency parsing. In Human Language Technologies: Pe Annual Conference of the North American Chapter of the Association for Computational Linguistics, p. 742–750, Los Angeles, California, June 2010.

Yoav Goldberg and Joakim Nivre. Training deterministic parsers with non-deterministic oracles. Transactions of the Association for Computational Linguistics, 1(0):403–414, October 2013. ISSN 2307-387X.

Yoav Goldberg, Kai Zhao and Liang Huang. Efficient implementation of beam-search incremental parsers. In Proc. of the 51st Annual Meeting of the Association for Computational Linguistics (volume 2: Short Papers), p. 628–633, Sofia, Bulgaria, August 2013.

Christoph Goller and Andreas Küchler. Learning task-dependent distributed representations by backpropagation through structure. In In Proc. of the ICNN-96, p. 347–352. IEEE, 1996.

Hila Gonen and Yoav Goldberg. Semi supervised preposition-sense disambiguation using multilingual data. In Proc. of COLING, the 26th International Conference on Computational Linguistics: Technical Papers, p. 2718–2729, Osaka, Japan, December 2016. The COLING 2016 Organizing Committee. http://aclweb.org/anthology/C16-1256.

Joshua Goodman. A bit of progress in language modeling. CoRR, cs.CL/0108005, 2001. http://arxiv.org/abs/cs.CL/0108005 DOI: 10.1006/csla.2001.0174.

Stephan Gouws, Yoshua Bengio and Greg Corrado. BilBOWA: Fast bilingual distributed representations without word alignments. In Proc. of the 32nd International Conference on Machine Learning, p. 748–756, 2015.

A. Graves. Supervised Sequence Labelling with Recurrent Neural Networks. Ph.D. thesis, Technische Universität München, 2008. DOI: 10.1007/978-3-642-24797-2.

Alex Graves, Greg Wayne and Ivo Danihelka. Neural turing machines. CoRR, abs/1410.5401, 2014. http://arxiv.org/abs/1410.5401.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman and Phil Blunsom. Learning to transduce with unbounded memory. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama and R. Garnett, Eds., Advances in Neural Information Processing Systems 28, p. 1828–1836.

Curran Associates, Inc., 2015. http://papers.nips.cc/paper/5648-learning-to-transducewith-unbounded-memory.pdf.

Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink and Jürgen Schmidhuber. LSTM: A search space odyssey. arXiv:1503.04069 [cs], March 2015. DOI: 10.1109/tnnls.2016.2582924.

Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In International Conference on Artificial Intelligence and Statistics, p. 297–304, 2010.

Zellig Harris. Distributional structure. Word, 10(23):146–162, 1954. DOI: 10.1080/00437956.1954.11659520.

Kazuma Hashimoto, Makoto Miwa, Yoshimasa Tsuruoka and Takashi Chikayama. Simple customization of recursive neural networks for semantic relation classification. In Proc. of the Conference on Empirical Methods in Natural Language Processing, p. 1372–1376, Seattle, Washington. Association for Computational Linguistics, October 2013.

Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. arXiv:1502.01852 [cs], February 2015. DOI: 10.1109/iccv.2015.123.

Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. Deep residual learning for image recognition. In Pe IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016. DOI: 10.1109/cvpr.2016.90.

Matthew Henderson, Blaise Thomson and Steve Young. Deep neural network approach for the dialog state tracking challenge. In Proc. of the SIGDIAL Conference, p. 467–471, Metz, France. Association for Computational Linguistics, August 2013.

Karl Moritz Hermann and Phil Blunsom. The role of syntax in vector space models of compositional semantics. In Proc. of the 51st Annual Meeting of the Association for Computational Linguistics (volume 1: Long Papers), p. 894–904, Sofia, Bulgaria, August 2013.

Karl Moritz Hermann and Phil Blunsom. Multilingual models for compositional distributed semantics. In Proc. of the 52nd Annual Meeting of the Association for Computational Linguistics (volume 1: Long Papers), p. 58–68, Baltimore, Maryland, June 2014. DOI: 10.3115/v1/p14-1006.

Salah El Hihi and Yoshua Bengio. Hierarchical recurrent neural networks for long-term dependencies. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, Eds., Advances in Neural Information Processing Systems 8, p. 493–499. MIT Press, 1996.

Felix Hill, Kyunghyun Cho, Sebastien Jean, Coline Devin and Yoshua Bengio. Embedding word similarity with neural machine translation. arXiv:1412.6448 [cs], December 2014.

Geoffrey E. Hinton, J. L. McClelland and D. E. Rumelhart. Distributed representations. In D. E. Rumelhart, J. L. McClelland, et al., Eds., Parallel Distributed Processing: Volume 1: Foundations, p. 77–109. MIT Press, Cambridge, 1987.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. arXiv:1207.0580 [cs], July 2012.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural Computation, 9(8): 1735–1780, 1997. DOI: 10.1162/neco.1997.9.8.1735.

Julia Hockenmaier. Data and Models for Statistical Parsing with Combinatory Categorial Grammar. Ph.D. thesis, University of Edinburgh, 2003. DOI: 10.3115/1073083.1073139.

Kurt Hornik, Maxwell Stinchcombe and Halbert White. Multilayer feedforward networks are universal approximators. Neural Networks, 2(5):359–366, 1989. ISSN 0893-6080. DOI: 10.1016/0893-6080(89)90020-8.

Dirk Hovy, Stephen Tratz and Eduard Hovy. What's in a preposition? dimensions of sense disambiguation for an interesting word class. In Coling Posters, p. 454–462, Beijing, China, August 2010. Coling 2010 Organizing Committee. http://www.aclweb.org/anthology/C10-2052.

(Kenneth) Ting-Hao Huang, Francis Ferraro, Nasrin Mostafazadeh, Ishan Misra, Aishwarya Agrawal, Jacob Devlin, Ross Girshick, Xiaodong He, Pushmeet Kohli, Dhruv Batra, Lawrence C. Zitnick, Devi Parikh, Lucy Vanderwende, Michel Galley and Margaret Mitchell. Visual storytelling. In Proc. of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, p. 1233–1239, 2016. http://aclweb.org/anthology/N16-1147 DOI: 10.18653/v1/N16-1147.

Liang Huang, Suphan Fayong and Yang Guo. Structured perceptron with inexact search. In Proc. of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, p. 142–151, 2012. http://aclweb.org/anthology/N12-1015.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167 [cs], February 2015.

Ozan Irsoy and Claire Cardie. Opinion mining with deep recurrent neural networks. In Proc. of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), p. 720–728, Doha, Qatar. Association for Computational Linguistics, October 2014. DOI: 10.3115/v1/d14-1080.

Mohit Iyyer, Jordan Boyd-Graber, Leonardo Claudino, Richard Socher and Hal Daumé III. A neural network for factoid question answering over paragraphs. In Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP), p. 633–644, Doha, Qatar. Association for Computational Linguistics, October 2014a. DOI: 10.3115/v1/d14-1070.

Mohit Iyyer, Peter Enns, Jordan Boyd-Graber and Philip Resnik. Political ideology detection using recursive neural networks. In Proc. of the 52nd Annual Meeting of the Association for Computational Linguistics (volume 1: Long Papers), p. 1113–1122, Baltimore, Maryland, June 2014b. DOI: 10.3115/v1/p14-1105.

Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber and Hal Daumé III. Deep unordered composition rivals syntactic methods for text classification. In Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (volume 1: Long Papers), p. 1681–1691, Beijing, China, July 2015. DOI: 10.3115/v1/p15-1162.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic and Yoshua Bengio. On using very large target vocabulary for neural machine translation. In Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (volume 1: Long Papers), p. 1–10, 2015. http://aclweb.org/anthology/P15-1001 DOI: 10.3115/v1/P15-1001.

Frederick Jelinek and Robert Mercer. Interpolated estimation of Markov source parameters from sparse data. In Workshop on Pattern Recognition in Practice, 1980.

Rie Johnson and Tong Zhang. Effective use of word order for text categorization with convolutional neural networks. In Proc. of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, p. 103–112, Denver, Colorado, 2015. DOI: 10.3115/v1/n15-1011.

Aravind K. Joshi and Bangalore Srinivas. Disambiguation of super parts of speech (or supertags): Allnost parsing. In COLING Volume 1: The 15th International Conference on Computational Linguistics, 1994. http://aclweb.org/anthology/C94-1024 DOI: 10.3115/991886.991912.

Armand Joulin, Edouard Grave, Piotr Bojanowski and Tomas Mikolov. Bag of tricks for efficient text classification. CoRR, abs/1607.01759, 2016. http://arxiv.org/abs/1607.01759.

Rafal Jozefowicz, Wojciech Zaremba and Ilya Sutskever. An empirical exploration of recurrent network architectures. In Proc. of the 32nd International Conference on Machine Learning (ICML-15), p. 2342–2350, 2015.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer and Yonghui Wu. Exploring the limits of language modeling. arXiv:1602.02410 [cs], February 2016.

Daniel Jurafsky and James H. Martin. Speech and Language Processing, 2nd ed. Prentice Hall, 2008.

Nal Kalchbrenner, Edward Grefenstette and Phil Blunsom. A convolutional neural network for modelling sentences. In Proc. of the 52nd Annual Meeting of the Association for Computational Linguistics (volume 1: Long Papers), p. 655–665, Baltimore, Maryland, June 2014. DOI: 10.3115/v1/p14-1062.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aäron van den Oord, Alex Graves and Koray Kavukcuoglu. Neural machine translation in linear time. CoRR, abs/1610.10099, 2016. http://arxiv.org/abs/1610.10099.

Katharina Kann and Hinrich Schütze. Proc. of the 14th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology, chapter MED: The LMU System for the SIGMORPHON 2016 Shared Task on Morphological Reinflection, p. 62–70. Association for Computational Linguistics, 2016. http://aclweb.org/anthology/W16-2010 DOI: 10.18653/v1/W16-2010.

Anjuli Kannan, Karol Kurach, Sujith Ravi, Tobias Kaufmann, Andrew Tomkins, Balint Miklos, Greg Corrado, Laszlo Lukacs, Marina Ganea, Peter Young and Vivek Ramavajjala. Smart reply: Automated response suggestion for email. In Proc. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), 2016. https://arxiv.org/pdf/1606.04870.pdf DOI: 10.1145/2939672.2939801.

Andrej Karpathy and Fei-Fei Li. Deep visual-semantic alignments for generating image descriptions. In IEEE Conference on Computer Vision and Pattern Recognition, CVPR, p.3128–3137, Boston, MA, June 7–12, 2015. DOI: 10.1109/cvpr.2015.7298932.

Andrej Karpathy, Justin Johnson and Fei-Fei Li. Visualizing and understanding recurrent networks. arXiv:1506.02078 [cs], June 2015.

Douwe Kiela and Stephen Clark. A systematic study of semantic vector space model parameters. In Workshop on Continuous Vector Space Models and their Compositionality, 2014. DOI: 10.3115/v1/w14-1503.

Yoon Kim. Convolutional neural networks for sentence classification. In Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP), p. 1746–1751, Doha, Qatar. Association for Computational Linguistics, October 2014. DOI: 10.3115/v1/d14-1181.

Yoon Kim, Yacine Jernite, David Sontag and Alexander M. Rush. Character-aware neural language models. arXiv:1508.06615 [cs, stat], August 2015.

Diederik Kingma and Jimmy Ba. ADAM: A method for stochastic optimization. arXiv: 1412.6980 [cs], December 2014.

Eliyahu Kiperwasser and Yoav Goldberg. Easy-first dependency parsing with hierarchical tree LSTMs. Transactions of the Association of Computational Linguistics (volume 4, Issue 1), p. 445–461, 2016a. http://aclweb.org/anthology/Q16-1032.

Eliyahu Kiperwasser and Yoav Goldberg. Simple and accurate dependency parsing using bidirectional LSTM feature representations. Transactions of the Association of Computational Linguistics (volume 4, Issue 1), p. 313–327, 2016b. http://aclweb.org/anthology/Q16-1023.

Karin Kipper, Hoa T. Dang and Martha Palmer. Class-based construction of a verb lexicon. In AAAI/IAAI, p. 691–696, 2000.

Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba and Sanja Fidler. Skip-thought vectors. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., Advances in Neural Information Processing Systems 28, p. 3294–3302. Curran Associates, Inc., 2015. http://papers.nips.cc/paper/5950-skip-thought-vectors.pdf.

Sigrid Klerke, Yoav Goldberg and Anders Søgaard. Improving sentence compression by learning to predict gaze. In Proc. of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, p. 1528–1533, 2016. http://aclweb.org/anthology/N16-1179 DOI: 10.18653/v1/N16-1179.

Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In Acoustics, Speech, and Signal Processing, ICASSP-95, International Conference on, volume 1, p. 181–184, May 1995. DOI: 10.1109/ICASSP.1995.479394.

Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In Proc. of MT Summit, volume 5, p. 79–86, 2005.

Philipp Koehn. Statistical Machine Translation. Cambridge University Press, 2010. DOI: 10.1017/cbo9780511815829.

Terry Koo and Michael Collins. Efficient third-order dependency parsers. In Proc. of the 48th Annual Meeting of the Association for Computational Linguistics, p. 1–11, 2010. http://aclweb.org/anthology/P10-1001.

Moshe Koppel, Jonathan Schler and Shlomo Argamon. Computational methods in authorship attribution. Journal of the American Society for information Science and Technology, 60(1):9–26, 2009. DOI: 10.1002/asi.20961.

Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou and K. Q. Weinberger, Eds., Advances in Neural Information Processing Systems 25, p. 1097–1105. Curran Associates, Inc., 2012. DOI: 10.1007/978-3-319-46654-5_20.

R. A. Kronmal and A. V. Peterson, Jr. On the alias method for generating random variables from a discrete distribution. Pe American Statistician, 33:214–218, 1979. DOI: 10.2307/2683739.

Sandra Kübler, Ryan McDonald and Joakim Nivre. Dependency Parsing. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2008. DOI: 10.2200/s00169ed1v01y200901hlt002.

Taku Kudo and Yuji Matsumoto. Fast methods for Kernel-based text analysis. In Proc. of the 41st Annual Meeting on Association for Computational Linguistics (volume 1), p. 24–31, Stroudsburg, PA, 2003. DOI: 10.3115/1075096.1075100.

John Lafferty, Andrew McCallum and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In Proc. of ICML, 2001.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami and Chris Dyer. Neural architectures for named entity recognition. In Proc. of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, p. 260–270, 2016. http://aclweb.org/anthology/N16-1030 DOI: 10.18653/v1/N16-1030.

Phong Le and Willem Zuidema. The inside-outside recursive neural network model for dependency parsing. In Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP), p. 729–739, Doha, Qatar. Association for Computational Linguistics, October 2014. DOI: 10.3115/v1/d14-1081.

Phong Le and Willem Zuidema. The forest convolutional network: Compositional distributional semantics with a neural chart and without binarization. In Proc. of the Conference on Empirical Methods in Natural Language Processing, p. 1155–1164, Lisbon, Portugal. Association for Computational Linguistics, September 2015. DOI: 10.18653/v1/d15-1137.

Quoc V. Le, Navdeep Jaitly and Geoffrey E. Hinton. A simple way to initialize recurrent networks of rectified linear units. arXiv:1504.00941 [cs], April 2015.

Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time-series. In M. A. Arbib, Ed., The Handbook of Brain Peory and Neural Networks. MIT Press, 1995.

Yann LeCun, Leon Bottou, G. Orr and K. Muller. Efficient BackProp. In G. Orr and Muller K, Eds., Neural Networks: Tricks of the Trade. Springer, 1998a. DOI: 10.1007/3-540-49430-8_2.

Yann LeCun, Leon Bottou, Yoshua Bengio and Patrick Haffner. Gradient based learning applied to pattern recognition. Proc. of the IEEE, 86(11):2278–2324, November 1998b.

Yann LeCun and F. Huang. Loss functions for discriminative training of energy-based models. In Proc. of AISTATS, 2005.

Yann LeCun, Sumit Chopra, Raia Hadsell, M. Ranzato and F. Huang. A tutorial on energy-based learning. Predicting Structured Data, 1:0, 2006.

Geunbae Lee, Margot Flowers and Michael G. Dyer. Learning distributed representations of conceptual knowledge and their application to script-based story processing. In Connectionist Natural Language Processing, p. 215–247. Springer, 1992. DOI: 10.1007/978-94-011-2624-3_11.

Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. Neural Networks, 6(6):861–867, 1993. ISSN 0893-6080. http://www.sciencedirect.com/science/article/pii/S0893608005801315 DOI: 10.1016/S0893-6080(05)80131-5.

Omer Levy and Yoav Goldberg. Dependency-based word embeddings. In Proc. of the 52nd Annual Meeting of the Association for Computational Linguistics (volume 2: Short Papers), p. 302–308, Baltimore, Maryland, June 2014. DOI: 10.3115/v1/p14-2050.

Omer Levy and Yoav Goldberg. Linguistic regularities in sparse and explicit word representations. In Proc. of the 18th Conference on Computational Natural Language Learning, p. 171–180. Association for Computational Linguistics, 2014. http://aclweb.org/anthology/W14-1618 DOI: 10.3115/v1/W14-1618.

Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., Advances in Neural Information Processing Systems 27, p. 2177–2185. Curran Associates, Inc., 2014.

Omer Levy, Yoav Goldberg and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. Transactions of the Association for Computational Linguistics, 3 (0):211–225, May 2015. ISSN 2307-387X.

Omer Levy, Anders Søgaard and Yoav Goldberg. A strong baseline for learning cross-lingual word embeddings from sentence alignments. In Proc. of the 15th Conference of the European Chapter of the Association for Computational Linguistics, 2017.

Mike Lewis and Mark Steedman. Improved CCG parsing with semi-supervised supertagging. Transactions of the Association for Computational Linguistics, 2(0):327–338, October 2014. ISSN 2307-387X.

Mike Lewis, Kenton Lee and Luke Zettlemoyer. LSTM CCG parsing. In Proc. of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, p. 221–231, 2016. http://aclweb.org/anthology/N16-1026 DOI: 10.18653/v1/N16-1026.

Jiwei Li, Rumeng Li and Eduard Hovy. Recursive deep models for discourse parsing. In Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP), p. 2061–2069, Doha, Qatar. Association for Computational Linguistics, October 2014. DOI: 10.3115/v1/d14-1220.

Jiwei Li, Thang Luong, Dan Jurafsky and Eduard Hovy. When are tree structures necessary for deep learning of representations? In Proc. of the Conference on Empirical

Methods in Natural Language Processing, p. 2304–2314. Association for Computational Linguistics, 2015. http://aclweb.org/anthology/D15-1278 DOI: 10.18653/v1/D15-1278.

Jiwei Li, Michel Galley, Chris Brockett, Georgios Spithourakis, Jianfeng Gao and Bill Dolan. A persona-based neural conversation model. In Proc. of the 54th Annual Meeting of the Association for Computational Linguistics (volume 1: Long Papers), p. 994–1003, 2016. http://aclweb.org/anthology/P16-1094 DOI: 10.18653/v1/P16-1094.

G. J. Lidstone. Note on the general case of the Bayes-Laplace formula for inductive or a posteriori probabilities. Transactions of the Faculty of Actuaries, 8:182–192, 1920.

Wang Ling, Chris Dyer, Alan W. Black and Isabel Trancoso. Two/too simple adaptations of Word2Vec for syntax problems. In Proc. of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, p. 1299–1304, Denver, Colorado, 2015a. DOI: 10.3115/v1/n15-1142.

Wang Ling, Chris Dyer, Alan W. Black, Isabel Trancoso, Ramon Fermandez, Silvio Amir, Luis Marujo and Tiago Luis. Finding function in form: Compositional character models for open vocabulary word representation. In Proc. of the Conference on Empirical Methods in Natural Language Processing, p. 1520–1530, Lisbon, Portugal. Association for Computational Linguistics, September 2015b. DOI: 10.18653/v1/d15-1176.

Tal Linzen, Emmanuel Dupoux and Yoav Goldberg. Assessing the ability of LSTMs to learn syntax-sensitive dependencies. Transactions of the Association for Computational Linguistics, 4: 521–535, 2016. ISSN 2307-387X. https://www.transacl.org/ojs/index.php/tacl/article/view/972.

Ken Litkowski and Orin Hargraves. The preposition project. In Proc. of the 2nd ACL-SIGSEM Workshop on the Linguistic Dimensions of Prepositions and Peir Use in Computational Linguistics Formalisms and Applications, p. 171–179, 2005.

Ken Litkowski and Orin Hargraves. SemEval-2007 task 06: Word-sense disambiguation of prepositions. In Proc. of the 4th International Workshop on Semantic Evaluations, p. 24–29, 2007. DOI: 10.3115/1621474.1621479.

Yang Liu, Furu Wei, Sujian Li, Heng Ji, Ming Zhou and Houfeng Wang. A dependency-based neural network for relation classification. In Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (volume 2: Short Papers), p. 285–290, Beijing, China, July 2015. DOI: 10.3115/v1/p15-2047.

Minh-Thang Luong, Hieu Pham and Christopher D. Manning. Effective approaches to attention-based neural machine translation. arXiv:1508.04025 [cs], August 2015.

Minh-Thang Luong, Quoc V. Le, Ilya Sutskever, Oriol Vinyals and Lukasz Kaiser. Multi-task sequence to sequence learning. In Proc. of ICLR, 2016.

Ji Ma, Yue Zhang and Jingbo Zhu. Tagging the web: Building a robust web tagger with neural network. In Proc. of the 52nd Annual Meeting of the Association for Computational Linguistics (volume 1: Long Papers), p. 144–154, Baltimore, Maryland, June 2014. DOI: 10.3115/v1/p14-1014.

Mingbo Ma, Liang Huang, Bowen Zhou and Bing Xiang. Dependency-based convolutional neural networks for sentence embedding. In Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (volume 2: Short Papers), p. 174–179, Beijing, China, July 2015. DOI: 10.3115/v1/p15-2029.

Xuezhe Ma and Eduard Hovy. End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. In Proc. of the 54th Annual Meeting of the Association for Computation-

al Linguistics (volume 1: Long Papers), p. 1064–1074, Berlin, Germany, August 2016. http://www.aclweb.org/anthology/P16-1101 DOI: 10.18653/v1/p16-1101.

Christopher Manning and Hinrich Schütze. Foundations of Statistical Natural Language Processing. MIT Press, 1999.

Christopher Manning, Prabhakar Raghavan and Hinrich Schütze. Introduction to Information Retrieval. Cambridge University Press, 2008. DOI: 10.1017/cbo9780511809071.

Junhua Mao, Wei Xu, Yi Yang, Jiang Wang and Alan L. Yuille. Explain images with multimodal recurrent neural networks. CoRR, abs/1410.1090, 2014. http://arxiv.org/abs/1410.1090.

Ryan McDonald, Koby Crammer and Fernando Pereira. Online large-margin training of dependency parsers. In Proc. of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05), p. 91–98, 2005. http://aclweb.org/anthology/P05-1012 DOI: 10.3115/1219840.1219852.

Ryan McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith B. Hall, Slav Petrov, Hao Zhang, Oscar Täckström, Claudia Bedini, Núria Bertomeu Castelló and Jungmee Lee. Universal dependency annotation for multilingual parsing. In ACL (2), p. 92–97, 2013.

Tomáš Mikolov. Statistical language models based on neural networks. Ph.D. thesis, Brno University of Technology, 2012.

Tomáš Mikolov, Martin Karafiát, Lukas Burget, Jan Cernocky and Sanjeev Khudanpur. Recurrent neural network based language model. In INTERSPEECH, 11th Annual Conference of the International Speech Communication Association, p. 1045–1048, Makuhari, Chiba, Japan, September 26–30, 2010.

Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Honza Černocky and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In Acoustics, Speech and Signal Processing (ICASSP), IEEE International Conference on, p. 5528–5531, 2011. DOI: 10.1109/icassp.2011.5947611.

Tomáš Mikolov, Kai Chen, Greg Corrado and Jeffrey Dean. Efficient estimation of word representations in vector space. arXiv:1301.3781 [cs], January 2013.

Tomáš Mikolov, Quoc V. Le and Ilya Sutskever. Exploiting similarities among languages for machine translation. CoRR, abs/1309.4168, 2013. http://arxiv.org/abs/1309.4168.

Tomáš Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani and K. Q. Weinberger, Eds., Advances in Neural Information Processing Systems 26, p. 3111–3119. Curran Associates, Inc., 2013.

Tomáš Mikolov, Wen-tau Yih and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In Proc. of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, p. 746–751, 2013. http://aclweb.org/anthology/N13-1090.

Tomáš Mikolov, Armand Joulin, Sumit Chopra, Michael Mathieu and Marc'Aurelio Ranzato. Learning longer memory in recurrent neural networks. arXiv:1412.7753 [cs], December 2014.

Scott Miller, Jethran Guinness and Alex Zamanian. Name tagging with word clusters and discriminative training. In Proc. of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL, 2004. http://aclweb.org/anthology/N04-1043.

Andriy Mnih and Koray Kavukcuoglu. Learning word embeddings efficiently with noise-contrastive estimation. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani,

and K. Q. Weinberger, Eds., Advances in Neural Information Processing Systems 26, p. 2265–2273. Curran Associates, Inc., 2013.

Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. In John Langford and Joelle Pineau, Eds., Proc. of the 29th International Conference on Machine Learning (ICML-12), p. 1751–1758, New York, NY, July 2012. Omnipress.

Mehryar Mohri, Afshin Rostamizadeh and Ameet Talwalkar. Foundations of Machine Learning. MIT Press, 2012.

Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In Robert G. Cowell and Zoubin Ghahramani, Eds., Proc. of the 10th International Workshop on Artificial Intelligence and Statistics, p. 246–252, 2005. http://www.iro.umontreal.ca/~lisa/pointeurs/hierarchical-nnlm-aistats05.pdf.

Nikola Mrkšić, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gasic, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen and Steve Young. Multi-domain dialog state tracking using recurrent neural networks. In Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (volume 2: Short Papers), p. 794–799, Beijing, China. Association for Computational Linguistics, July 2015. DOI: 10.3115/v1/p15-2130.

Masami Nakamura and Kiyohiro Shikano. A study of English word category prediction based on neural networks. The Journal of the Acoustical Society of America, 84(S1):S60–S61, 1988. DOI: 10.1121/1.2026400.

R. Neidinger. Introduction to automatic differentiation and MATLAB object-oriented programming. SIAM Review, 52(3):545–563, January 2010. ISSN 0036-1445. DOI: 10.1137/080743627.

Y. Nesterov. A method of solving a convex programming problem with convergence rate O (1/k2). In Soviet Mathematics Doklady, 27:372–376, 1983.

Y. Nesterov. Introductory Lectures on Convex Optimization. Kluwer Academic Publishers, 2004. DOI: 10.1007/978-1-4419-8853-9.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta and Pengcheng Yin. DyNet: The dynamic neural network toolkit. CoRR, abs/1701.03980, 2017. http://arxiv.org/abs/1701.03980.

Thien Huu Nguyen and Ralph Grishman. Event detection and domain adaptation with convolutional neural networks. In Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (volume 2: Short Papers), p. 365–371, Beijing, China, July 2015. DOI: 10.3115/v1/p15-2060.

Joakim Nivre. Algorithms for deterministic incremental dependency parsing. Computational Linguistics, 34(4):513–553, December 2008. ISSN 0891-2017, 1530-9312. DOI: 10.1162/coli.07-056-R1-07-027.

Joakim Nivre, Željko Agić, Maria Jesus Aranzabe, Masayuki Asahara, Aitziber Atutxa, Miguel Ballesteros, John Bauer, Kepa Bengoetxea, Riyaz Ahmad Bhat, Cristina Bosco, Sam Bowman, Giuseppe G. A. Celano, Miriam Connor, Marie-Catherine de Marneffe, Arantza Diaz de Ilarraza, Kaja Dobrovoljc, Timothy Dozat, Tomaž Erjavec, Richárd Farkas, Jennifer Foster, Daniel Galbraith, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Yoav Goldberg,

Berta Gonzales, Bruno Guillaume, Jan Hajič, Dag Haug, Radu Ion, Elena Irimia, Anders Johannsen, Hiroshi Kanayama, Jenna Kanerva, Simon Krek, Veronika Laippala, Alessandro Lenci, Nikola Ljubešić, Teresa Lynn, Christopher Manning, Cătălina Mărănduc, David Mareček, Héctor Martínez Alonso, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Anna Missilä, Verginica Mititelu, Yusuke Miyao, Simonetta Montemagni, Shunsuke Mori, Hanna Nurmi, Petya Osenova, Lilja Øvrelid, Elena Pascual, Marco Passarotti, Cenel-Augusto Perez, Slav Petrov, Jussi Piitulainen, Barbara Plank, Martin Popel, Prokopis Prokopidis, Sampo Pyysalo, Loganathan Ramasamy, Rudolf Rosa, Shadi Saleh, Sebastian Schuster, Wolfgang Seeker, Mojgan Seraji, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Kiril Simov, Aaron Smith, Jan Štěpánek, Alane Suhr, Zsolt Szántó, Takaaki Tanaka, Reut Tsarfaty, Sumire Uematsu, Larraitz Uria, Viktor Varga, Veronika Vincze, Zdeněk Žabokrtský, Daniel Zeman and Hanzhi Zhu. Universal dependencies 1.2, 2015. http://hdl.handle.net/11234/1-1548 LINDAT/CLARIN digital library at Institute of Formal and Applied Linguistics, Charles University in Prague.

Chris Okasaki. Purely Functional Data Structures. Cambridge University Press, Cambridge, UK, June 1999. DOI: 10.1017/cbo9780511530104.

Mitchell P. Marcus, Beatrice Santorini and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. Computational Linguistics, 19(2), June 1993, Special Issue on Using Large Corpora: II, 1993. http://aclweb.org/anthology/J93-2004.

Martha Palmer, Daniel Gildea and Nianwen Xue. Semantic Role Labeling. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2010. DOI: 10.1093/oxfordhb/9780199573691.013.023.

Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. Foundation and Trends in Information Retrieval, 2:1–135, 2008. DOI: 10.1561/1500000011.

Ankur P. Parikh, Oscar Täckström, Dipanjan Das and Jakob Uszkoreit. A decomposable attention model for natural language inference. In Proc. of EMNLP, 2016. DOI: 10.18653/v1/d16-1244.

Razvan Pascanu, Tomas Mikolov and Yoshua Bengio. On the difficulty of training recurrent neural networks. arXiv:1211.5063 [cs], November 2012.

Ellie Pavlick, Pushpendre Rastogi, Juri Ganitkevitch, Benjamin Van Durme and Chris Callison-Burch. PPDB 2.0: Better paraphrase ranking, fine-grained entailment relations, word embeddings, and style classification. In Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (volume 2: Short Papers), p. 425–430. Association for Computational Linguistics, 2015. http://aclweb.org/anthology/P15-2070 DOI: 10.3115/v1/P15-2070.

Wenzhe Pei, Tao Ge and Baobao Chang. An effective neural network model for graph-based dependency parsing. In Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (volume 1: Long Papers), p. 313–322, Beijing, China, July 2015. DOI: 10.3115/v1/p15-1031.

Joris Pelemans, Noam Shazeer and Ciprian Chelba. Sparse non-negative matrix language modeling. Transactions of the Association of Computational Linguistics, 4(1): 329–342, 2016. http://aclweb.org/anthology/Q16-1024.

Jian Peng, Liefeng Bo and Jinbo Xu. Conditional neural fields. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, Eds., Advances in Neural Information Processing Systems 22, p. 1419–1427. Curran Associates, Inc., 2009.

Jeffrey Pennington, Richard Socher and Christopher Manning. GloVe: global vectors for word representation. In Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP), p. 1532–1543, Doha, Qatar. Association for Computational Linguistics, October 2014. DOI: 10.3115/v1/d14-1162.

Vu Pham, Christopher Kermorvant and Jérôme Louradour. Dropout improves recurrent neural networks for handwriting recognition. CoRR, abs/1312.4569, 2013. http://arxiv.org/abs/1312.4569 DOI: 10.1109/icfhr.2014.55.

Barbara Plank, Anders Søgaard and Yoav Goldberg. Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. In Proc. of the 54th Annual Meeting of the Association for Computational Linguistics (volume 2: Short Papers), p. 412–418. Association for Computational Linguistics, 2016. http://aclweb.org/anthology/P16-2067 DOI: 10.18653/v1/P16-2067.

Jordan B. Pollack. Recursive distributed representations. Artificial Intelligence, 46:77–105, 1990. DOI: 10.1016/0004-3702(90)90005-k.

B. T. Polyak. Some methods of speeding up the convergence of iteration methods. USSR Computational Mathematics and Mathematical Physics, 4(5):1–17, 1964. ISSN 0041-5553. DOI: 10.1016/0041-5553(64)90137-5.

Qiao Qian, Bo Tian, Minlie Huang, Yang Liu, Xuan Zhu and Xiaoyan Zhu. Learning tag embeddings and tag-specific composition functions in recursive neural network. In Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (volume 1: Long Papers), p. 1365–1374, Beijing, China, July 2015. DOI: 10.3115/v1/p15-1132.

Lev Ratinov and Dan Roth. Proc. of the 13th Conference on Computational Natural Language Learning (CoNLL-2009), chapter Design Challenges and Misconceptions in Named Entity Recognition, p. 147–155. Association for Computational Linguistics, 2009. http://aclweb.org/anthology/W09-1119.

Ronald Rosenfeld. A maximum entropy approach to adaptive statistical language modeling. Computer, Speech and Language, 10:187–228, 1996. Longe version: Carnegie Mellon Technical Report CMU-CS-94-138. DOI: 10.1006/csla.1996.0011.

Stéphane Ross and J. Andrew Bagnell. Efficient reductions for imitation learning. In Proc. of the 13th International Conference on Artificial Intelligence and Statistics, p. 661–668, 2010.

Stéphane Ross, Geoffrey J. Gordon and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In Proc. of the 14th International Conference on Artificial Intelligence and Statistics, p. 627–635, 2011.

David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams. Learning representations by back-propagating errors. Nature, 323(6088):533–536, October 1986. DOI: 10.1038/323533a0.

Ivan A. Sag, Thomas Wasow and Emily M. Bender. Syntactic Peory, 2nd ed., CSLI Lecture Note 152, 2003.

Magnus Sahlgren. The distributional hypothesis. Italian Journal of Linguistics, 20(1):33–54, 2008.

Nathan Schneider, Vivek Srikumar, Jena D. Hwang and Martha Palmer. A hierarchy with, of, and for preposition supersenses. In Proc. of the 9th Linguistic Annotation Workshop, p. 112–123, 2015. DOI: 10.3115/v1/w15-1612.

Nathan Schneider, Jena D. Hwang, Vivek Srikumar, Meredith Green, Abhijit Suresh, Kathryn Conger, Tim O'Gorman and Martha Palmer. A corpus of preposition supersenses. In

Proc. of the 10th Linguistic Annotation Workshop, 2016. DOI: 10.18653/v1/w16-1712.

Bernhard Schölkopf. The kernel trick for distances. In T. K. Leen, T. G. Dietterich, and V. Tresp, Eds., Advances in Neural Information Processing Systems 13, p. 301–307. MIT Press, 2001. http://papers.nips.cc/paper/1862-the-kernel-trick-for-distances.pdf.

M. Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing, 45(11):2673–2681, November 1997. ISSN 1053-587X. DOI: 10.1109/78.650093.

Holger Schwenk, Daniel Dchelotte and Jean-Luc Gauvain. Continuous space language models for statistical machine translation. In Proc. of the COLING/ACL on Main Conference Poster Sessions, p. 723–730. Association for Computational Linguistics, 2006. DOI: 10.3115/1273073.1273166.

Rico Sennrich and Barry Haddow. Proc. of the 1st Conference on Machine Translation: Volume 1, Research Papers, chapter Linguistic Input Features Improve Neural Machine Translation, p. 83–91. Association for Computational Linguistics, 2016. http://aclweb.org/anthology/W16-2209 DOI: 10.18653/v1/W16-2209.

Rico Sennrich, Barry Haddow and Alexandra Birch. Neural machine translation of rare words with subword units. In Proc. of the 54th Annual Meeting of the Association for Computational Linguistics (volume 1: Long Papers), p. 1715–1725, 2016a. http://aclweb.org/anthology/P16-1162 DOI: 10.18653/v1/P16-1162.

Rico Sennrich, Barry Haddow and Alexandra Birch. Improving neural machine translation models with monolingual data. In Proc. of the 54th Annual Meeting of the Association for Computational Linguistics (volume 1: Long Papers), p. 86–96. Association for Computational Linguistics, 2016b. http://aclweb.org/anthology/P16-1009 DOI: 10.18653/v1/P16-1009.

Shai Shalev-Shwartz and Shai Ben-David. Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press, 2014. DOI: 10.1017/cbo97811072980191.

John Shawe-Taylor and Nello Cristianini. Kernel Methods for Pattern Analysis. Cambridge University Press, Cambridge, UK, June 2004. DOI: 10.4018/9781599040424.ch001.

Q. Shi, J. Petterson, G. Dror, J. Langford, A. J. Smola, A. Strehl and V. Vishwanathan. Hash kernels. In Artificial Intelligence and Statistics AISTATS'09, Florida, April 2009.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

Noah A. Smith. Linguistic Structure Prediction. Synthesis Lectures on Human Language Technologies. Morgan & Claypool, May 2011. DOI: 10.2200/s00361ed1v01y201105hlt013.

Richard Socher. Recursive Deep Learning For Natural Language Processing and Computer Vision. Ph.D. thesis, Stanford University, August 2014.

Richard Socher, Christopher Manning and Andrew Ng. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In Proc. of the Deep Learning and Unsupervised Feature Learning Workshop of {NIPS}, p. 1–9, 2010.

Richard Socher, Cliff Chiung-Yu Lin, Andrew Y. Ng and Christopher D. Manning. Parsing natural scenes and natural language with recursive neural networks. In Lise Getoor and Tobias Scheffer, Eds., Proc. of the 28th International Conference on Machine Learning, *ICML*, p. 129–136, Bellevue, Washington, June 28–July 2, Omnipress, 2011.

[1] *Шай Шалев-Шварц, Шай Бен-Давид*. Идеи машинного обучения. М.: ДМК Пресс, 2019.

Richard Socher, Brody Huval, Christopher D. Manning and Andrew Y. Ng. Semantic compositionality through recursive matrix-vector spaces. In Proc. of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, p. 1201–1211, Jeju Island, Korea. Association for Computational Linguistics, July 2012.

Richard Socher, John Bauer, Christopher D. Manning and Andrew Y. Ng. Parsing with compositional vector grammars. In Proc. of the 51st Annual Meeting of the Association for Computational Linguistics (volume 1: Long Papers), p. 455–465, Sofia, Bulgaria, August 2013a.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In Proc. of the 2013 Conference on Empirical Methods in Natural Language Processing, p. 1631–1642, Seattle, Washington. Association for Computational Linguistics, October 2013b.

Anders Søgaard. Semi-Supervised Learning and Domain Adaptation in Natural Language Processing. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2013. DOI: 10.2200/s00497ed1v01y201304hlt021.

Anders Søgaard and Yoav Goldberg. Deep multi-task learning with low level tasks supervised at lower layers. In Proc. of the 54th Annual Meeting of the Association for Computational Linguistics (volume 2: Short Papers), p. 231–235, 2016. http://aclweb.org/anthology/P16-2038 DOI: 10.18653/v1/P16-2038.

Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Margaret Mitchell, Jian-Yun Nie, Jianfeng Gao and Bill Dolan. A neural network approach to context-sensitive generation of conversational responses. In Proc. of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, p. 196–205, Denver, Colorado, 2015. DOI: 10.3115/v1/n15-1020.

Vivek Srikumar and Dan Roth. An inventory of preposition relations. arXiv:1305.5785, 2013a.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15:1929–1958, 2014. http://jmlr.org/papers/v15/srivastava14a.html.

E. Strubell, P. Verga, D. Belanger and A. McCallum. Fast and accurate sequence labeling with iterated dilated convolutions. ArXiv e-prints, February 2017.

Martin Sundermeyer, Ralf Schlüter and Hermann Ney. LSTM neural networks for language modeling. In INTERSPEECH, 2012.

Martin Sundermeyer, Tamer Alkhouli, Joern Wuebker and Hermann Ney. Translation modeling with bidirectional recurrent neural networks. In Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP), p. 14–25, Doha, Qatar. Association for Computational Linguistics, October 2014. DOI: 10.3115/v1/d14-1003.

Ilya Sutskever, James Martens and Geoffrey E. Hinton. Generating text with recurrent neural networks. In Proc. of the 28th International Conference on Machine Learning (ICML-11), p. 1017–1024, 2011. DOI: 10.1109/icnn.1993.298658.

Ilya Sutskever, James Martens, George Dahl and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Proc. of the 30th International Conference on Machine Learning (ICML-13), p. 1139–1147, 2013.

Ilya Sutskever, Oriol Vinyals and Quoc V. V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., Advances in Neural Information Processing Systems 27, p. 3104–3112. Curran Associates, Inc., 2014.

Kai Sheng Tai, Richard Socher and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (volume 1: Long Papers), p. 1556–1566, Beijing, China, July 2015. DOI: 10.3115/v1/p15-1150.

Akihiro Tamura, Taro Watanabe and Eiichiro Sumita. Recurrent neural networks for word alignment model. In Proc. of the 52nd Annual Meeting of the Association for Computational Linguistics (volume 1: Long Papers), p. 1470–1480, Baltimore, Maryland, June 2014. DOI: 10.3115/v1/p14-1138.

Duyu Tang, Bing Qin and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. In Proc. of the Conference on Empirical Methods in Natural Language Processing, p. 1422–1432. Association for Computational Linguistics, 2015. http://aclweb.org/anthology/D15-1167 DOI: 10.18653/v1/D15-1167.

Matus Telgarsky. Benefits of depth in neural networks. *arXiv:1602.04485 [cs, stat]*, February 2016.

Robert Tibshirani. Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society, Series B, 58:267–288, 1994. DOI: 10.1111/j.1467-9868.2011.00771.x.

T. Tieleman and G. Hinton. Lecture 6.5 – RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

Joseph Turian, Lev-Arie Ratinov and Yoshua Bengio. Word representations: A simple and general method for semi-supervised learning. In Proc. of the 48th Annual Meeting of the Association for Computational Linguistics, p. 384–394, 2010. http://aclweb.org/anthology/P10-1040.

Peter D. Turney. Mining the web for synonyms: PMI-IR vs. LSA on TOEFL. In ECML, 2001. DOI: 10.1007/3-540-44795-4_42.

Peter D. Turney and Patrick Pantel. From frequency to meaning: Vector space models of semantics. Journal of Artificial Intelligence Research, 37(1):141–188, 2010.

Jakob Uszkoreit, Jay Ponte, Ashok Popat and Moshe Dubiner. Large scale parallel document mining for machine translation. In Proc. of the 23rd International Conference on Computational Linguistics (Coling 2010), p. 1101–1109, Organizing Committee, 2010. http://aclweb.org/anthology/C10-1124.

Tim Van de Cruys. A neural network approach to selectional preference acquisition. In Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP), p. 26–35, Doha, Qatar. Association for Computational Linguistics, October 2014. DOI: 10.3115/v1/d14-1004.

Ashish Vaswani, Yinggong Zhao, Victoria Fossum and David Chiang. Decoding with largescale neural language models improves translation. In Proc. of the Conference on Empirical Methods in Natural Language Processing, p. 1387–1392, Seattle, Washington. Association for Computational Linguistics, October 2013.

Ashish Vaswani, Yinggong Zhao, Victoria Fossum and David Chiang. Decoding with large-scale neural language models improves translation. In Proc. of the Conference on Empirical Methods in Natural Language Processing, p. 1387–1392. Association for Computational Linguistics, 2013. http://aclweb.org/anthology/D13-1140.

Ashish Vaswani, Yonatan Bisk, Kenji Sagae and Ryan Musa. Supertagging with LSTMs. In Proc. of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, p. 232–237. Association for Computational Linguistics, 2016. http://aclweb.org/anthology/N16-1027 DOI: 10.18653/v1/N16-1027.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever and Geoffrey Hinton. Grammar as a foreign language. arXiv:1412.7449 [cs, stat], December 2014.

Oriol Vinyals, Alexander Toshev, Samy Bengio and Dumitru Erhan. Show and tell: A neural image caption generator. In IEEE Conference on Computer Vision and Pattern Recognition, CVPR, p. 3156–3164, Boston, MA, June 7–12, 2015. DOI: 10.1109/cvpr.2015.7298935.

Stefan Wager, Sida Wang and Percy S Liang. Dropout training as adaptive regularization. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., Advances in Neural Information Processing Systems 26, p. 351–359. Curran Associates, Inc., 2013.

Mengqiu Wang and Christopher D. Manning. Effect of non-linear deep architecture in sequence labeling. In IJCNLP, p. 1285–1291, 2013.

Peng Wang, Jiaming Xu, Bo Xu, Chenglin Liu, Heng Zhang, Fangyuan Wang and Hongwei Hao. Semantic clustering and convolutional neural network for short text categorization. In Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (volume 2: Short Papers), p. 352–357, Beijing, China, July 2015a. DOI: 10.3115/v1/p15-2058.

Xin Wang, Yuanchao Liu, Chengjie Sun, Baoxun Wang and Xiaolong Wang. Predicting polarities of tweets by composing word embeddings with long short-term memory. In Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (volume 1: Long Papers), p. 1343–1353, Beijing, China, July 2015b. DOI: 10.3115/v1/p15-1130.

Taro Watanabe and Eiichiro Sumita. Transition-based neural constituent parsing. In Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (volume 1: Long Papers), p. 1169–1179, Beijing, China, July 2015. DOI: 10.3115/v1/p15-1113.

K. Weinberger, A. Dasgupta, J. Attenberg, J. Langford and A. J. Smola. Feature hashing for large scale multitask learning. In International Conference on Machine Learning, 2009. DOI: 10.1145/1553374.1553516.

David Weiss, Chris Alberti, Michael Collins and Slav Petrov. Structured training for neural network transition-based parsing. In Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (volume 1: Long Papers), p. 323–333, Beijing, China, July 2015. DOI: 10.3115/v1/p15-1032.

P. J. Werbos. Backpropagation through time: What it does and how to do it. Proc. of the IEEE, 78(10):1550–1560, 1990. ISSN 0018-9219. DOI: 10.1109/5.58337.

Jason Weston, Antoine Bordes, Oksana Yakhnenko and Nicolas Usunier. Connecting language and knowledge bases with embedding models for relation extraction. In Proc. of the Conference on Empirical Methods in Natural Language Processing, p. 1366–1371, Seattle, Washington. Association for Computational Linguistics, October 2013.

Philip Williams, Rico Sennrich, Matt Post and Philipp Koehn. Syntax-based Statistical Machine Translation. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2016. DOI: 10.2200/s00716ed1v04y201604hlt033.

Sam Wiseman and Alexander M. Rush. Sequence-to-sequence learning as beam-search optimization. In Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics, 2016. DOI: 10.18653/v1/d16-1137.

Sam Wiseman, M. Alexander Rush and M. Stuart Shieber. Learning global features for coreference resolution. In Proc. of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, p. 994–1004, 2016. http://aclweb.org/anthology/N16-1114 DOI: 10.18653/v1/N16-1114.

Yijun Xiao and Kyunghyun Cho. Efficient character-level document classification by combining convolution and recurrent layers. CoRR, abs/1602.00367, 2016. http://arxiv.org/abs/1602.00367.

Wenduan Xu, Michael Auli and Stephen Clark. CCG supertagging with a recurrent neural network. In Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (volume 2: Short Papers), p. 250–255, Beijing, China, July 2015. DOI: 10.3115/v1/p15-2041.

Wenpeng Yin and Hinrich Schütze. Convolutional neural network for paraphrase identification. In Proc. of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, p. 901–911, Denver, Colorado, 2015. DOI: 10.3115/v1/n15-1091.

Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *ICLR*, 2016.

Wojciech Zaremba, Ilya Sutskever and Oriol Vinyals. Recurrent neural network regularization. arXiv:1409.2329 [cs], September 2014.

Matthew D. Zeiler. ADADELTA: An adaptive learning rate method. arXiv:1212.5701 [cs], December 2012.

Daojian Zeng, Kang Liu, Siwei Lai, Guangyou Zhou and Jun Zhao. Relation classification via convolutional deep neural network. In Proc. of COLING, the 25th International Conference on Computational Linguistics: Technical Papers, page 2335–2344, Dublin, Ireland, Dublin City University and Association for Computational Linguistics, August 2014.

Hao Zhang and Ryan McDonald. Generalized higher-order dependency parsing with cube pruning. In Proc. of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, p. 320–331. Association for Computational Linguistics, 2012. http://aclweb.org/anthology/D12-1030.

Tong Zhang. Statistical behavior and consistency of classification methods based on convex risk minimization. Pe Annals of Statistics, 32:56–85, 2004. DOI: 10.1214/aos/1079120130.

Xiang Zhang, Junbo Zhao and Yann LeCun. Character-level convolutional networks for text classification. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., Advances in Neural Information Processing Systems 28, p. 649–657. Curran Associates, Inc., 2015. http://papers.nips.cc/paper/5782-character-level-convolutional-networks-for-text-classification.pdf.

Xingxing Zhang, Jianpeng Cheng and Mirella Lapata. Dependency parsing as head selection. *CoRR*, abs/1606.01280, 2016. http://arxiv.org/abs/1606.01280.

Yuan Zhang and David Weiss. Stack-propagation: Improved representation learning for syntax. In Proc. of the 54th Annual Meeting of the Association for Computational

Linguistics (volume 1: Long Papers), p. 1557–1566, 2016. http://aclweb.org/anthology/P16-1147 DOI: 10.18653/v1/P16-1147.

Hao Zhou, Yue Zhang, Shujian Huang and Jiajun Chen. A neural probabilistic structured-prediction model for transition-based dependency parsing. In Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (volume 1: Long Papers), p. 1213–1222, Beijing, China, July 2015. DOI: 10.3115/v1/p15-1117.

Chenxi Zhu, Xipeng Qiu, Xinchi Chen and Xuanjing Huang. A re-ranking model for dependency parser with recursive convolutional neural network. In Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (volume 1: Long Papers), p. 1159–1168, Beijing, China, July 2015a. DOI: 10.3115/v1/p15-1112.

Xiaodan Zhu, Parinaz Sobhani and Hongyu Guo. Long short-term memory over tree structures. March 2015b.

Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society, Series B, 67:301–320, 2005. DOI: 10.1111/j.1467-9868.2005.00503.x.

Книги издательства «ДМК Пресс» можно заказать
в торгово-издательском холдинге «Планета Альянс» наложенным платежом,
выслав открытку или письмо по почтовому адресу:
115487, г. Москва, 2-й Нагатинский пр-д, д. 6А.
При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.
Желательно также указать свой телефон и электронный адрес.
Эти книги вы можете заказать и в интернет-магазине: **www.a-planeta.ru**.
Оптовые закупки: тел. **(499) 782-38-89**.
Электронный адрес: **books@alians-kniga.ru**.

Йоав Гольдберг

Нейросетевые методы в обработке естественного языка

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Перевод *Слинкин А. А.*
Корректор *Синяева Г. И.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Формат 70×100 1/16.
Гарнитура «PT Serif». Печать офсетная.
Усл. печ. л. 22,91. Тираж 200 экз.

Веб-сайт издательства: **www.dmkpress.com**