Об авторах

Саваш Йылдырым (Savaş Yıldırım) окончил факультет информационных технологий Стамбульского технического университета и имеет степень доктора философии в области обработки естественного языка (NLP). В настоящее время является адъюнкт-профессором Стамбульского университета Билги, Турция, и приглашенным исследователем в Университете Райерсона, Канада. Активный лектор и исследователь с более чем 20-летним опытом преподавания курсов по машинному обучению, глубокому обучению и NLP. Внес значительный вклад в турецкое сообщество NLP, разработав большое количество приложений и ресурсов с открытым исходным кодом. Он также предоставляет комплексные консультации компаниям, занимающимся ИИ, по их проектам, связанным с исследованиями. В свободное время пишет книги и снимает короткометражные фильмы, а также занимается йогой.

Прежде всего я хотел бы поблагодарить мою дорогую подругу Айлин Октай за ее постоянную поддержку и терпение на протяжении длительного процесса написания этой книги. Я также хотел бы поблагодарить своих коллег из факультета информационных технологий Стамбульского университета Билги за их поддержку.

Мейсам Асгари-Ченаглу (Meysam Asgari-Chenaghlu) – менеджер по ИИ в компании Carbon Consulting, а также кандидат наук в Тебризском университете. Он был консультантом ведущих телекоммуникационных и банковских компаний Турции. Работал над различными проектами, включая понимание естественного языка и семантический поиск.

Прежде всего я хотел бы поблагодарить мою любящую и терпеливую жену Наржес Никзад-Хасмахи за ее поддержку и понимание. Я также хотел бы поблагодарить моего отца за его поддержку; пусть его душа упокоится с миром. Большое спасибо Carbon Consulting и моим коллегам.

О рецензенте

Александр Афанасьев – инженер-программист с 14-летним опытом работы в различных отраслях и на разных должностях. В настоящее время Александр является независимым подрядчиком, который реализует идеи в области компьютерного зрения, NLP и создания передовых систем сбора данных в области анализа киберугроз. Раньше Александр уже выступал рецензентом книги *Selenium Testing Cookbook* от издательства Packt. Помимо повседневной работы, он активно участвует в деятельности Stack Overflow и GitHub.

Я хочу поблагодарить авторов этой книги за их усердную работу и представленные в этой книге новаторские идеи; благодарю замечательную команду редакторов и координаторов с их отличными коммуникативными навыками и мою семью, которая всегда поддерживала мои идеи и мою работу.

Оглавление

Об авторах ... 5

О рецензенте ... 6

Предисловие ... 11

Для кого эта книга ... 11
Какие темы охватывает эта книга ... 11
Как получить максимальную отдачу от этой книги ... 12
Скачивание исходного кода примеров ... 13
Видеоролики Code in Action ... 13
Условные обозначения и соглашения, принятые в книге ... 13
Список опечаток ... 14
Нарушение авторских прав ... 14

ЧАСТЬ I. ПОСЛЕДНИЕ РАЗРАБОТКИ В ОБЛАСТИ NLP, ПОДГОТОВКА РАБОЧЕЙ СРЕДЫ И ПРИЛОЖЕНИЕ HELLO WORLD ... 15

Глава 1. От последовательности слов к трансформерам ... 17

Технические требования ... 18
Эволюция подходов NLP в направлении трансформеров ... 18
Что такое дистрибутивная семантика? ... 21
Использование глубокого обучения ... 26
Обзор архитектуры трансформеров ... 37
Трансформеры и перенос обучения ... 46
Заключение ... 48
Дополнительная литература ... 48

Глава 2. Знакомство с трансформерами на практике ... 49

Технические требования ... 50
Установка библиотеки `Transformer` с Anaconda ... 51
Работа с языковыми моделями и токенизаторами ... 57

Работа с моделями, предоставленными сообществом ... 59
Сравнительное тестирование и наборы данных ... 62
Тестирование быстродействия и использования памяти ... 74
Заключение ... 77

ЧАСТЬ II. МОДЕЛИ-ТРАНСФОРМЕРЫ – ОТ АВТОЭНКОДЕРОВ К АВТОРЕГРЕССИИ ... 79

Глава 3. Языковые модели на основе автоэнкодеров ... 81

Технические требования ... 82
BERT – одна из языковых моделей на основе автоэнкодера ... 82
Обучение автоэнкодерной языковой модели для любого языка ... 86
Как поделиться моделями с сообществом ... 97
Обзор других моделей с автоэнкодером ... 98
Использование алгоритмов токенизации ... 104
Заключение ... 115

Глава 4. Авторегрессивные и другие языковые модели ... 116

Технические требования ... 117
Работа с языковыми моделями AR ... 117
Работа с моделями Seq2Seq ... 122
Обучение авторегрессивной языковой модели ... 127
Генерация текста с использованием авторегрессивных моделей ... 132
Тонкая настройка резюмирования и машинного перевода с помощью `simpletransformers` ... 135
Заключение ... 138
Дополнительная литература ... 138

Глава 5. Тонкая настройка языковых моделей для классификации текста ... 139

Технические требования ... 140
Введение в классификацию текста ... 140
Тонкая настройка модели BERT для двоичной классификации с одним предложением ... 141
Обучение модели классификации с помощью PyTorch ... 148
Тонкая настройка BERT для многоклассовой классификации с пользовательскими наборами данных ... 152
Тонкая настройка BERT для регрессии пар предложений ... 158

Использование `run_glue.py` для тонкой настройки моделей 163
Заключение 164

Глава 6. Тонкая настройка языковых моделей для классификации токенов 165

Технические требования 166
Введение в классификацию токенов 166
Тонкая настройка языковых моделей для NER 171
Ответы на вопросы с использованием классификации токенов 179
Заключение 187

Глава 7. Представление текста 188

Технические требования 188
Введение в представление предложений 189
Эксперимент по выявлению семантического сходства с FLAIR 198
Кластеризация текста с помощью Sentence-BERT 204
Семантический поиск с помощью Sentence-BERT 209
Заключение 213
Дополнительная литература 214

ЧАСТЬ III. ДОПОЛНИТЕЛЬНЫЕ ТЕМЫ 215

Глава 8. Работа с эффективными трансформерами 217

Технические требования 218
Обзор эффективных, легких и быстрых трансформеров 218
Способы уменьшения размера модели 220
Работа с эффективным самовниманием 226
Заключение 246
Дополнительная литература 247

Глава 9. Многоязычные и кросс-языковые модели 248

Технические требования 249
Моделирование языка перевода и обмен знаниями между языками 249
XLM и mBERT 251
Задачи выявления кросс-языкового сходства 256
Кросс-языковая классификация 263
Кросс-языковое обучение без подготовки 268
Фундаментальные ограничения многоязычных моделей 271
Заключение 274
Дополнительная литература 274

Глава 10. Трансформерная модель как самостоятельная служба 275

Технические требования 276
Запуск службы трансформерной модели с fastAPI 276
Докеризация API 279
Создание службы модели с использованием TFX 280
Нагрузочное тестирование службы с помощью Locust 282
Заключение 286
Дополнительные источники информации 286

Глава 11. Визуализация внимания и отслеживание экспериментов 287

Технические требования 288
Интерпретация механизма внимания 288
Многоуровневая визуализация потоков внимания с помощью BertViz 294
Заключение 312
Дополнительная литература 313

Предметный указатель 314

Предисловие

Мы стали свидетелями больших изменений в обработке естественного языка (natural language processing, NLP), которые случились за последние 20 лет. За это время мы испробовали в деле разные подходы и, наконец, вступили в новую эру доминирования удивительной нейросетевой архитектуры трансформеров. Эта архитектура глубокого обучения унаследовала многие методы своих предшественников. В число данных методов входит *контекстное векторное представление слов* (contextual word embedding), *многопоточное самовнимание* (multi-head self-attention), *позиционное кодирование* (positional encodings), *параллелизируемые архитектуры* (parallelizable architectures), *сжатие моделей* (model compression), *перенос обучения* (transfer learning) и *кросс-языковые модели* (cross-lingual model). Архитектура трансформеров начала свое развитие с различных нейронных подходов к NLP, постепенно превратилась в архитектуру кодировщика-декодера на основе механизма внимания и продолжает развиваться в этом направлении по сей день. Сейчас в литературе появляются описания новых успешных вариантов этой архитектуры. Появились отличные модели, которые используют только кодировщик, такие как BERT, или только декодер, такие как GPT.

На протяжении всей книги мы будем возвращаться к упомянутым методам, а работа с трансформерами не составит для нас труда благодаря библиотеке Transformers от сообщества Hugging Face. Мы предложим пошаговые решения широкого спектра задач в области NLP, начиная от обобщения и заканчивая ответами на вопросы. Мы покажем, что с помощью нейросетевых трансформеров можно достичь самых современных результатов.

Для кого эта книга

Эта книга предназначена для исследователей, работающих в области глубокого обучения, практических специалистов по NLP, преподавателей машинного обучения / NLP и студентов, которые хотят начать свой путь в машинное обучение с обработки естественного языка. Знание машинного обучения хотя бы на начальном уровне и хорошие навыки программирования на Python помогут вам извлечь максимальную пользу из этой книги.

Какие темы охватывает эта книга

В *главе 1* дается краткое введение в историю NLP и проводится сравнение традиционных методов и моделей глубокого обучения, таких как CNN, RNN и LSTM, и моделей нейросетевых трансформеров.

В *главе 2* мы более подробно расскажем об использовании трансформеров. Для токенизаторов и таких моделей, как BERT, мы приведем практические примеры.

В *главе 3* вы узнаете о том, как обучать языковые модели с автоэнкодером на любом заданном языке с нуля, включая предварительное обучение и обучение моделей конкретным задачам.

В *главе 4* представлены теоретические основы авторегрессионных языковых моделей и рассказано о том, как предварительно обучить их на собственном корпусе. Вы узнаете, как выполнить предварительное обучение любой языковой модели, такой как GPT-2, на собственном тексте и использовать модель в различных задачах, таких как генерация текста на естественном языке.

В *главе 5* вы узнаете, как настроить предварительно обученную модель для классификации текста и для любой последующей задачи классификации, такой как анализ тональности или многоклассовая классификация.

В *главе 6* рассказывается о тонкой настройке языковых моделей для задач классификации токенов, таких как NER, POS-теги и ответы на вопросы.

В *главе 7* вы узнаете о методах представления текста и о том, как эффективно использовать трансформерную архитектуру, особенно для задач без обучающего набора, таких как кластеризация, семантический поиск и выделение тем текстов.

В *главе 8* показано, как создавать эффективные модели с помощью дистилляции, усечения и дискретизации. Вы узнаете об эффективных разреженных трансформерах, таких как Linformer и BigBird, и о том, как с ними работать.

В *главе 9* вы узнаете о предварительном обучении многоязыковой и межъязыковой модели и различиях между многоязычным и межъязычным предварительным обучением. Кроме того, в этой главе мы рассматриваем модели причинно-следственной связи и языка перевода.

В *главе 10* подробно описано, как выполнять приложения NLP на основе трансформеров в средах, где доступны два вида процессоров – центральный и графический. Здесь также будет описано использование платформы TensorFlow Extended (TFX) для развертывания машинного обучения.

В *главе 11* представлены две различные технические концепции: визуализация механизма внимания и отслеживание эксперимента. Мы испытаем их в действии на примере сложных инструментов, таких как exBERT и BertViz.

Как получить максимальную отдачу от этой книги

Чтобы получить максимальную отдачу от этой книги, читателю необходимо владеть навыками программирования на языке Python, знать основы NLP и глубокого обучения и понимать, как работают глубокие нейронные сети.

> **Важное примечание**
>
> Весь код в этой книге соответствует версии Python 3.6, поскольку некоторые библиотеки для версии Python 3.9 находятся в стадии разработки.

Программное и аппаратное обеспечение	Необходимая операционная система
Transformers	Windows, macOS, Linux
TensorFlow и PyTorch	Windows, macOS, Linux
Python 3.6x	Windows, macOS, Linux
Jupyter Notebook	Windows, macOS, Linux
Google Colaboratory	Windows, macOS, Linux
Docker	Windows, macOS, Linux
Locust.io	Windows, macOS, Linux
Git	Windows, macOS, Linux

Если вы используете цифровую версию этой книги, мы рекомендуем скачать код из репозитория книги на GitHub по приведенной ниже ссылке. Это поможет вам избежать любых потенциальных ошибок, связанных с копированием и вставкой кода.

Скачивание исходного кода примеров

Вы можете скачать файлы примеров кода для этой книги с GitHub по адресу https://github.com/PacktPublishing/Mastering-Transformers. Если есть обновление кода, оно появится в репозитории GitHub.

Видеоролики Code in Action

Видеоролики *Code in Action* для этой книги (на английском языке) можно посмотреть на YouTube по адресу https://bit.ly/3i4vFzJ.

Условные обозначения и соглашения, принятые в книге

В книге используются следующие типографские соглашения.

Курсив – используется для смыслового выделения важных положений, новых терминов, имен команд и утилит, а также слов и предложений на естественном языке.

`Моноширинный шрифт` – применяется для листингов программ, а также в обычном тексте для обозначения имен переменных, функций, типов, объектов, баз данных, переменных среды, операторов, ключевых слов и других программных конструкций и элементов исходного кода.

`Моноширинный полужирный шрифт` – используется для обозначения команд или фрагментов текста, которые пользователь должен ввести дословно без изменений, а также в листингах программ, если необходимо обратить особое внимание на фрагмент кода.

`Моноширинный курсив` – применяется для обозначения в исходном коде или в командах шаблонных меток-заполнителей, которые должны быть заменены соответствующими контексту реальными значениями.

> **Советы или важные примечания**
>
> Представляют собой текст, помещенный в рамку.

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 данной 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 веб-сайта, 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.

Часть I

Последние разработки в области NLP, подготовка рабочей среды и приложение Hello World

В этой части книги вы познакомитесь с архитектурой трансформеров на вводном уровне. Вы создадите свою первую программу `hello-world`, загрузив предоставленные сообществом предварительно обученные языковые модели и запустив соответствующий код с графическим процессором или без него. Но перед этим мы подробно расскажем об установке и использовании библиотек `tensorflow`, `pytorch`, `conda`, `transformers` и `sentenceTransformers`.

Этот раздел состоит из следующих глав:

- главы 1 «От последовательности слов к трансформерам»;
- главы 2 «Знакомство с трансформерами на практике».

1

От последовательности слов к трансформерам

В этой главе мы расскажем о том, что изменилось в *обработке естественного языка* (natural language processing, NLP) за два десятилетия. За это время мы испробовали в деле разные подходы и, наконец, вступили в новую эру доминирования удивительной нейросетевой архитектуры трансформеров. Все подходы по-своему помогают нам представить слова и документы для решения задач NLP. *Дистрибутивная семантика* описывает значение слова или документа при помощи векторного представления, рассматривая дистрибутивные вхождения в коллекции статей. Векторные представления используются для решения многих задач в процессах обработки естественного языка, как связанных, так и не связанных с машинным обучением. В течение многих лет для задач генерации текстов на естественном языке широко использовались языковые модели на основе *n*-грамм. Однако у этих традиционных подходов есть много недостатков, которые мы будем обсуждать на протяжении всей главы.

Далее мы обсудим классические архитектуры *глубокого обучения* (deep learning, DL), такие как *рекуррентные нейронные сети* (recurrent neural network, RNN), *нейронные сети с прямым распространением* (feed-forward neural network, FFNN) и *сверточные нейронные сети* (convolutional neural network, CNN). Благодаря использованию этих архитектур удалось повысить быстродействие приложений в области NLP и преодолеть ограничения традиционных подходов. Однако и у этих моделей есть свои проблемы и недостатки. В последнее время большой интерес вызывают модели-трансформеры, демонстрирующие удивительную эффективность во всех задачах NLP, от классификации до генерации текста. Однако главный успех трансформеров заключается в том, что

они значительно повысили быстродействие многоязычных и многопотоковых процессов NLP, а также одноязычных и однопотоковых задач. Благодаря архитектуре трансформеров стало намного проще применять в NLP *перенос обучения* (transfer learning, TL), назначение которого – сделать модели повторно применяемыми для разных задач или разных языков.

Мы начнем с механизма внимания, а затем кратко обсудим архитектуру трансформеров и различия между предыдущими моделями NLP. Параллельно с теоретическими рассуждениями мы будем демонстрировать практические примеры на основе популярной среды разработки NLP. Для простоты будем использовать как можно более короткие ознакомительные примеры кода.

В этой главе мы рассмотрим следующие темы:

- эволюция подходов NLP в направлении трансформеров;
- понятие дистрибутивной семантики;
- использование глубокого обучения;
- обзорное знакомство с архитектурой трансформеров;
- использование переноса обучения совместно с трансформерами.

Технические требования

Для упражнений по программированию мы будем использовать Jupyter Notebook. Нам потребуется интепретатор Python версии 3.6.0 или новее, а также следующие пакеты, которые необходимо установить с помощью команды `pip install`:

- `sklearn`;
- `nltk` (версия 3.5.0);
- `gensim` (версия 3.8.3);
- `fasttext`;
- `keras` (версия 2.3.0 или новее);
- `Transformers` (версия 4.00 или новее).

Все блокноты Jupyter с упражнениями доступны на GitHub по адресу: https://github.com/PacktPublishing/Advanced-Natural-Language-Processing-with-Transformers/tree/main/CH01.

Для просмотра видеоролика Code in Action перейдите по ссылке: https://bit.ly/2UFPuVd.

Эволюция подходов NLP в направлении трансформеров

Мы стали свидетелями больших изменений в обработке естественного языка, которые случились за последние 20 лет. За это время мы испробовали в деле разные подходы и, наконец, вступили в новую эру доминирования удивительной нейросетевой архитектуры *трансформеров*. Эта архитектура глубокого обучения унаследовала многие методы своих предшественников. Ее эволюция началась с различных методов нейронного моделирования, постепенно перешла к архитектуре энкодера-декодера с механизмом внимания и продолжает развиваться. Архитектура трансформеров и ее вариации достигли успеха благодаря следующим разработкам минувшего десятилетия:

- контекстно-векторное представление слов;
- улучшенные алгоритмы токенизации для обработки невидимых или редких слов;
- использование дополнительных токенов запоминания в предложениях, таких как `Paragraph ID` в `Doc2vec` или токен классификации CLS в языковой модели BERT;
- механизмы внимания, которые устраняют необходимость кодировать всю информацию предложения в один вектор контекста;
- механизм многопоточного самовнимания;
- позиционное кодирование порядка слов;
- параллелизируемые архитектуры, ускоряющие обучение и точную настройку;
- сжатие моделей (дистилляция, дискретизация и т. д.);
- перенос обучения (многоязычное и многозадачное обучение).

В течение многих лет мы использовали традиционные подходы NLP, такие как *языковые модели n-грамм*, *модели извлечения информации на основе TF-IDF* и *матрицы терминов документа с позиционным кодированием*. Все эти подходы внесли большой вклад в решение различных задач NLP, таких как классификация последовательностей, генерация и понимание текстов на естественном языке и т. д.

С другой стороны, у этих традиционных методов NLP есть свои слабые стороны, например неспособность решить проблемы разреженности, представления невидимых слов, отслеживания протяженных зависимостей и др. Для устранения этих недостатков были разработаны подходы на основе DL, такие как:

- RNN;
- CNN;
- FFNN;
- несколько вариантов, объединяющих RNN, CNN и FFNN.

В 2013 году Word2vec – модель двухслойного кодировщика слов FFNN – решила проблему размерности, создавая короткие и плотные представления слов, которые называются *векторным представлением* (word embedding). Эта модель-предшественник позволяла генерировать быстрые и эффективные статические векторные представления слов. Она преобразовывала исходные текстовые данные в данные машинного обучения (точнее, *самообучения*) путем либо предсказания целевого слова с использованием контекста, либо предсказания соседних слов на основе скользящего окна. Создатели еще одной широко используемой и популярной модели – GloVe – утверждали, что модели, основанные на подсчете, могут работать лучше нейронных моделей. Она использует как глобальную, так и локальную статистику текстового корпуса, чтобы изучать векторы на основе статистики совпадения слов. Эта модель хорошо справляется с некоторыми синтаксическими и семантическими задачами, как показано на рис. 1.1. На рисунке хорошо видно, что смещения векторов определений помогают сформировать векторно-ориентированное восприятие. Мы можем сформировать обобщение гендерных отношений, которое является семантическим отношением смещения между мужчиной и женщиной (*мужчина* → *женщина*). Затем можем арифметически вычислить вектор термина *актриса*, сложив вектор тер-

мина *актер* и вычисленное ранее смещение. Точно так же мы можем исследовать синтаксические отношения, такие как формы множественного числа слов. Например, если известны векторы слов *актер*, *актеры* и *актриса*, мы можем вычислить вектор множественного числа женского рода (*актрисы*).

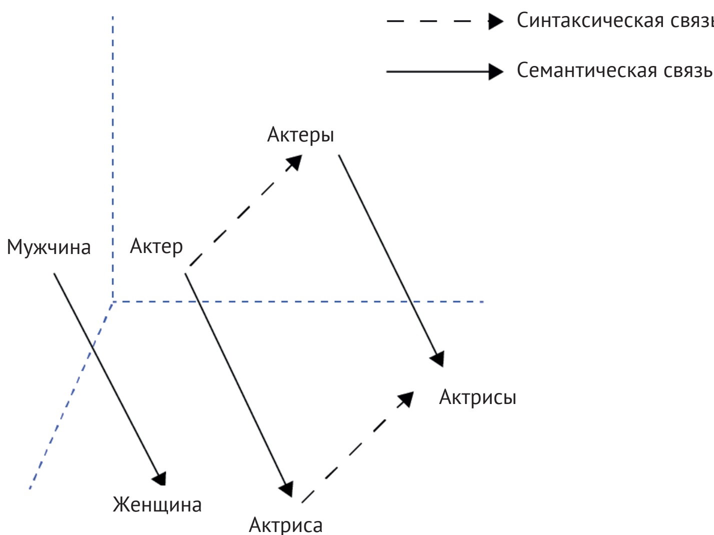

Рис. 1.1. Смещение векторов слов для извлечения отношений

В задачах преобразования последовательности в последовательность (sequence to sequence, seq2seq) в качестве кодировщиков и декодеров стали использовать рекуррентные и сверточные архитектуры, такие как RNN, CNN и с долгой краткосрочной памятью (long short-term memory, LSTM). Основная проблема, с которой столкнулись эти ранние модели, заключалась в многозначности слов. Смысл слов приходилось игнорировать, поскольку каждому слову назначалось одно фиксированное представление, что является особенно серьезной проблемой для многозначных слов и семантики предложений.

Следующим новаторским моделям нейронных сетей, таким как *универсальная языковая модель с тонкой настройкой* (universal language model finetuning, ULMFit) и *векторное представление на основе языковых моделей* (embeddings from language models, ELMo), удалось закодировать информацию на уровне предложений и наконец разрешить (хотя и не полностью) проблему многозначности, с которой не справлялись модели со статичными представлениями слов. Эти два важных подхода основаны на сетях LSTM и реализуют концепции предварительной тренировки и тонкой настройки. Они позволяют нам применять перенос обучения, используя модели, предварительно обученные общей задаче на огромных текстовых наборах данных.

Затем мы можем легко выполнить тонкую настройку, выполнив дообучение предварительно обученной сети на размеченных данных целевой задачи. В данном случае представления слов отличаются от традиционных векторов, поскольку каждое представление слова является функцией всего входного предложения. Современная архитектура трансформеров основана именно на этой идее.

Параллельно с эволюцией представлений слов развивалась идея механизма внимания, которая произвела сильное впечатление в области NLP и привела к значительным успехам, особенно в задачах типа seq2seq. Более ранние методы передавали последнее состояние (известное как *вектор контекста*, или *вектор смысла*), полученное из всей входной последовательности, в выходную последовательность без связывания или исключения. Механизм внимания способен построить более сложную модель, связав токены, определенные во входной последовательности, с конкретными токенами в выходной последовательности. Например, предположим, что у вас есть ключевая фраза *Government of Canada* (Правительство Канады) в исходном предложении, которое нужно перевести с английского на турецкий. В выходном предложении токен *Kanada Hükümeti* образует сильные связи с исходной фразой и более слабую связь с оставшимися словами в исходном предложении, как показано на рис. 1.2.

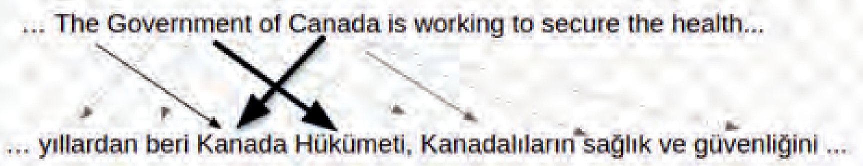

Рис. 1.2. Схематическое представление механизма внимания

Модели, использующие механизм внимания, более успешно решают задачи seq2seq, такие как перевод, ответы на вопросы и резюмирование текста.

В 2017 году была предложена успешная модель кодировщика-декодера на основе нейросети, относящейся к типу трансформеров. Ее архитектура основана на FFNN, но без использования рекуррентности RNN и применяет только механизмы внимания (Vaswani et al., *All you need is attention*, 2017). Модели, основанные на трансформерах, к настоящему времени преодолели многие трудности, с которыми сталкивались другие подходы, и стали новой ключевой парадигмой. На протяжении всей этой книги вы будете изучать, как работают модели на основе трансформеров.

Что такое дистрибутивная семантика?

Дистрибутивная семантика описывает значение слова в виде векторного представления, в первую очередь исследуя характеристики встречаемости, а не его словарные определения. Теория предполагает, что слова, встречающиеся вместе в одной и той же среде, имеют схожие значения. Впервые ее сформулировал ученый Харрис (*Distributional Structure Word*, 1954). Например, слова *собака* и *кошка* чаще всего встречаются в одном и том же контексте. Одним из преимуществ дистрибутивного подхода является возможность исследовать и отслеживать так называемые *лексико-семантические изменения* – семантическую эволюцию слов с течением времени и в разных областях.

Традиционные подходы на протяжении многих лет опирались на языковые модели *неупорядоченных наборов слов* (Bag of Words, BoW) и *n*-граммы для построения представления слов и предложений. В подходе BoW слова и доку-

менты представляются с помощью *прямого унитарного кодирования* (one-hot encoding), которое является разреженным способом представления, также известным как *модель векторного пространства* (Vector Space Model, VSM).

Классификация текста, выявление сходства слов, извлечение семантических отношений, устранение неоднозначности смысла слов – эти и многие другие задачи NLP решали с помощью методов унитарного кодирования в течение многих лет. В свою очередь, модели языка на основе *n*-грамм присваивают вероятности последовательностям слов, чтобы мы могли либо вычислить вероятность того, что последовательность принадлежит корпусу, либо сгенерировать случайную последовательность на основе данного корпуса.

Реализация BoW

BoW – это метод представления документов путем подсчета слов в них. Основная структура данной методики – это матрица документ–термин. Давайте рассмотрим простую реализацию BoW на языке Python. В следующем фрагменте кода показано, как построить матрицу терминов документа с помощью библиотеки Python `sklearn` для демонстрационного корпуса `toy_corpus`, состоящего из трех предложений:

```
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np
import pandas as pd
toy_corpus= ["the fat cat sat on the mat",
             "the big cat slept",
             "the dog chased a cat"]
vectorizer=TfidfVectorizer()
corpus_tfidf=vectorizer.fit_transform(toy_corpus)
print(f"The vocabulary size is \
{len(vectorizer.vocabulary_.keys())} ")
print(f"The document-term matrix shape is\
{corpus_tfidf.shape}")
df=pd.DataFrame(np.round(corpus_tfidf.toarray(),2))
df.columns=vectorizer.get_feature_names()
```

Результатом работы кода является матрица документ–термин, представленная на рис. 1.3. Размерность матрицы очень мала (3×10), но в реалистичном сценарии размеры матрицы могут быть довольно большими, например 10 тыс. × 10 млн.

```
The vocabulary size is 10
The document-term matrix shape is (3, 10)
```

	big	cat	chased	dog	fat	mat	on	sat	slept	the
0	0.00	0.25	0.00	0.00	0.42	0.42	0.42	0.42	0.00	0.49
1	0.61	0.36	0.00	0.00	0.00	0.00	0.00	0.00	0.61	0.36
2	0.00	0.36	0.61	0.61	0.00	0.00	0.00	0.00	0.00	0.36

Рис. 1.3. Матрица терминов документа

Данная матрица основана на подсчете, где значения ячеек сформированы в соответствии с весовой схемой «частота терминат – обратная частота документа» (term frequency-inverse document frequency, TF-IDF). Этот подход не учитывает положение слов. Поскольку порядок слов во многом определяет значение фразы, его игнорирование приводит к потере смысла. Это обычная проблема метода BoW, которая наконец решена с помощью механизма рекурсии в RNN и позиционного кодирования в трансформерах.

Каждый столбец в матрице обозначает вектор слова в словаре, а каждая строка обозначает вектор документа. Для вычисления сходства или несходства слов, а также документов могут применяться метрики семантического сходства. В большинстве случаев для улучшения представления документа мы используем биграммы, такие как `cat_sat` и `the_street`. Например, когда параметр `ngram_range=(1,2)` передается в `TfidfVectorizer`, он строит векторное пространство, содержащее как униграммы (`big`, `cat`, `dog`), так и биграммы (`big_cat`, `big_dog`). Такие модели также называют BoW-*n*-граммами, потому что они являются естественным продолжением BoW.

Если слово обычно используется в каждом документе, как, например, английский предлог *and*, его называют часто встречающимся, или высокочастотным. И наоборот, некоторые слова почти не встречаются в документах, поэтому их называют низкочастотными (или редкими) словами. Поскольку наличие в тексте высокочастотных и низкочастотных слов может помешать правильной работе модели, в качестве решения применяют TF-IDF, который является одним из наиболее важных и хорошо известных механизмов взвешивания.

Обратная частота документа (IDF) – это статистический вес для измерения важности слова в документе. Например, хотя слово *the* (определенный артикль) довольно часто встречается в английском языке, у него очень маленькая различающая способность, зато слово *chased* (гналась) может быть очень информативным и давать подсказки о теме текста. Это связано с тем, что часто используемые слова (стоп-слова, функциональные слова) имеют малую различающую способность при понимании документов.

Различимость терминов также зависит от предметной области – например, список статей про глубокое обучение, скорее всего, будет содержать слово «сеть» почти в каждом документе. IDF может уменьшить веса всех терминов, используя их *частоту документов* (document frequency, DF), которая вычисляется по количеству документов, включающих термин. *Частота термина* (term frequency, TF) – это исходное количество вхождений термина (слова) в документе.

Некоторые преимущества и недостатки модели BoW на основе TF-IDF перечислены в табл. 1.

Таблица 1. Преимущества и недостатки модели TF-IDF BoW

Преимущества	Недостатки
• простота реализации • результаты поддаются толкованию • адаптация к предметной области	• стремительный рост размерности • нет решения для невидимых слов • сложность выявления семантических отношений и синонимов • игнорируется порядок слов • медленно работает с большими словарями

Решение проблемы размерности

Для устранения размерности модели BoW широко используется *латентный семантический анализ* (Latent Semantic Analysis, LSA), позволяющий выявлять семантику в низкоразмерном пространстве. Это линейный метод, который фиксирует попарные корреляции между терминами. Вероятностные методы на основе LSA можно по-прежнему рассматривать как единый слой скрытых тематических переменных. Однако современные модели DL включают в себя несколько скрытых слоев с миллиардами параметров. Кроме того, модели на основе трансформеров показали, что они могут обнаруживать скрытые представления намного лучше, чем такие традиционные модели.

Когда мы решаем задачи *понимания естественного языка* (natural language understanding, NLU), традиционный процесс начинается с подготовительных шагов, таких как *токенизация*, *выделение морфологических основ* (stemming), *обнаружение именных словосочетаний* (noun phrase detection), *разбиение на части* (chunking), *удаление стоп-слов* и многого другого. После этого создается матрица документ–термин на основе какого-либо алгоритма взвешивания (самым популярным остается TF-IDF). Далее эта матрица служит входными табличными данными для процедуры *машинного обучения* (machine learning, ML), анализа тональности, сходства документов, кластеризации документов или оценки степени релевантности между запросом и документом. Аналогичным образом термины, представленные в виде матрицы, можно использовать для задачи классификации токенов, включая распознавание именованных объектов, извлечение семантических отношений и т. д.

Этап классификации включает в себя прямую реализацию таких алгоритмов машинного обучения на размеченных данных, как *машина опорных векторов* (support vector machine, SVM)[1], *случайный лес*, *логистика*, *наивный байесовский алгоритм* и *множественное обучение* (ускорение, или бэггинг). На практике реализация этой последовательности выглядит приблизительно как в следующем фрагменте кода:

```
from sklearn.pipeline import make_pipeline
from sklearn.svm import SVC
labels= [0,1,0]
clf = SVC()
clf.fit(df.to_numpy(), labels)
```

Как видно на примере этого кода, мы можем с легкостью использовать алгоритм подгонки модели (операцию `fit`) благодаря API `sklearn`. Чтобы применить обученную модель к обучающим данным, достаточно выполнить следующий код:

```
clf.predict(df.to_numpy())
Output: array([0, 1, 0])
```

Итак, переходим к следующему разделу!

[1] Категория нейросетей прямого распространения. – *Прим. перев.*

Моделирование языка и генерация текстов

Традиционные подходы к решению задачи генерации текстов на естественном языке основаны на использовании языковых моделей *n*-грамм. Это разновидности *марковского процесса*, представляющего собой стохастическую модель, в которой каждое слово (событие) зависит от подмножества предыдущих слов – *униграмм*, *биграмм* или *n-грамм*, определенных следующим образом:

- *униграмма* (все слова независимы и не образуют цепочки): оценивает вероятность появления слова в словаре, просто вычисляемую по его частоте вхождений в текст по отношению к общему количеству слов;
- *биграмма* (марковский процесс первого порядка): оценивает $P(word_i \mid word_{i-1})$ – вероятность $word_i$ в зависимости от $word_{i-1}$, которая просто вычисляется как отношение $P(word_i \mid word_{i-1})$ к $P(word_{i-1})$;
- *n-грамма* (марковский процесс *N*-го порядка): оценивает вероятность $P(word_i \mid word_0, \ldots, word_{i-1})$.

Рассмотрим простой пример реализации языковой модели с помощью библиотеки `Natural Language Toolkit` (NLTK). В следующей реализации мы обучаем *оцениватель максимального правдоподобия* (maximum likelihood estimator, MLE) с порядком $n = 2$. Мы можем выбрать любой порядок *n*-граммы, например $n = 1$ для униграмм, $n = 2$ для биграмм, $n = 3$ для триграмм и т. д.:

```
import nltk
from nltk.corpus import gutenberg
from nltk.lm import MLE
from nltk.lm.preprocessing import padded_everygram_pipeline
nltk.download('gutenberg')
nltk.download('punkt')
macbeth = gutenberg.sents('shakespeare-macbeth.txt')
model, vocab = padded_everygram_pipeline(2, macbeth)
lm=MLE(2)
lm.fit(model,vocab)
print(list(lm.vocab)[:10])
print(f"The number of words is {len(lm.vocab)}")
```

Пакет `nltk` сначала загружает корпус `gutenberg`, который содержит некоторые тексты из электронного текстового архива Project Gutenberg, размещенного по адресу https://www.gutenberg.org. Он также загружает токенизатор `punkt` для обработки пунктуации. Этот токенизатор делит необработанный текст на список предложений с помощью необучаемого алгоритма. Пакет `nltk` уже содержит предварительно обученную английскую модель токенизатора пунктуации для сокращенных слов и словосочетаний. Перед использованием его можно обучить пунктуации любого языка. В следующих главах мы покажем, как обучать различные и более эффективные токенизаторы для моделей на основе трансформеров. В нижеприведенном фрагменте кода мы используем уже обученную модель:

```
print(f"The frequency of the term 'Macbeth' is {lm.
counts['Macbeth']}")
print(f"The language model probability score of 'Macbeth' is
{lm.score('Macbeth')}")
print(f"The number of times 'Macbeth' follows 'Enter' is {lm.
counts[['Enter']]['Macbeth']} ")
print(f"P(Macbeth | Enter) is {lm.score('Macbeth',
['Enter'])}")
print(f"P(shaking | for) is {lm.score('shaking', ['for'])}")
```

Этот код выводит в терминал следующий результат:

```
The frequency of the term 'Macbeth' is 61
The language model probability score of 'Macbeth' is 0.00226
The number of times 'Macbeth' follows 'Enter' is 15
P(Macbeth | Enter) is 0.1875
P(shaking | for) is 0.0121
```

Языковая модель на основе *n*-грамм подсчитывает *n*-граммы и вычисляет условную вероятность генерации предложения. Строка `lm = MLE(2)` означает использование MLE, что дает максимально вероятное предложение для каждой вероятности токена. Следующий код генерирует случайное предложение из 10 слов с заданным начальным условием `<s>`:

```
lm.generate(10, text_seed=['<s>'], random_seed=42)
```

Результат выглядит следующим образом:

```
['My', 'Bosome', 'franchis', "'", 's', 'of', 'time', ',', 'We','are']
```

Мы можем указать конкретное начальное условие с помощью параметра `text_seed`, который заставляет результат генерации зависеть от предыдущего контекста. В нашем предыдущем примере предыдущий контекст – это `<s>`, который представляет собой специальный токен, указывающий на начало предложения.

Итак, мы рассмотрели принципы, заложенные в основу традиционных моделей NLP, и продемонстрировали очень простые примеры реализации этих моделей с помощью популярных фреймворков. Теперь настало время обсудить, как нейросетевые языковые модели сформировали сегодняшний ландшафт NLP и как они преодолевают ограничения традиционных моделей.

Использование глубокого обучения

NLP – одна из областей, где широко и успешно используются архитектуры глубокого обучения. На протяжении десятилетий мы были свидетелями успеха различных архитектур, особенно в области представления слов и предложений. Далее мы расскажем о наиболее популярных подходах на примере часто используемых фреймворков.

Векторное представление слов

Языковые модели на основе нейронных сетей эффективно решали проблемы представления признаков и языкового моделирования, поскольку стало воз-

можным обучать более сложную нейронную архитектуру на гораздо больших наборах данных для построения коротких и плотных представлений. Разработанная в 2013 году модель Word2vec, которая сегодня является популярным методом векторного представления слов, использовала простую и эффективную архитектуру для формирования высококачественных представлений непрерывных последовательностей. Она превзошла другие модели во множестве синтаксических и семантических языковых задач, таких как *анализ тональности*, *обнаружение перефразирования*, *извлечение отношений* и т. д. Другой ключевой фактор популярности модели – ее гораздо более низкая вычислительная сложность. Она максимизирует вероятность текущего слова с учетом всех окружающих контекстных слов или наоборот.

В следующем фрагменте кода показан пример обучения модели с векторным представлением слов на предложениях из пьесы «Макбет»:

```
from gensim.models import Word2vec
model = Word2vec(sentences=macbeth, size=100, window= 4, min_
count=10, workers=4, iter=10)
```

Этот код формирует представления слов с размерностью вектора `100` при помощи скользящего контекстного окна длиной `5`. Чтобы визуализировать векторные представления слов в трехмерном пространстве, нам нужно уменьшить размерность до `3`, применив *анализ главных компонентов* (principal component analysis, PCA), как показано в следующем фрагменте кода:

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import random
np.random.seed(42)
words=list([e for e in model.wv.vocab if len(e)>4])
random.shuffle(words)
words3d = PCA(n_components=3,random_state=42).fit_
transform(model.wv[words[:100]])
def plotWords3D(vecs, words, title):
...
plotWords3D(words3d, words, "Visualizing Word2vec Word
Embeddings using PCA")
```

Полученная визуализация представлена на рис. 1.4.

Взглянув на диаграмму распределения представлений, мы видим, что главные герои пьесы Шекспира – Макбет, Малкольм, Банко, Макдуф и другие – расположены близко друг к другу. Точно так же вспомогательные глаголы английского языка *shall*, *should* и *would* (*должен*, *следует* – перевод зависит от контекста) расположились рядом друг с другом в левом нижнем углу рис. 1.4. Используя смещение векторов, мы также можем уловить аналогию *man–woman* = *uncle–aunt* (*мужчина–женщина* = *дядя–тетя*). Более интересные визуальные примеры по этой теме можно найти в проекте по адресу https://projector.tensorflow.org/.

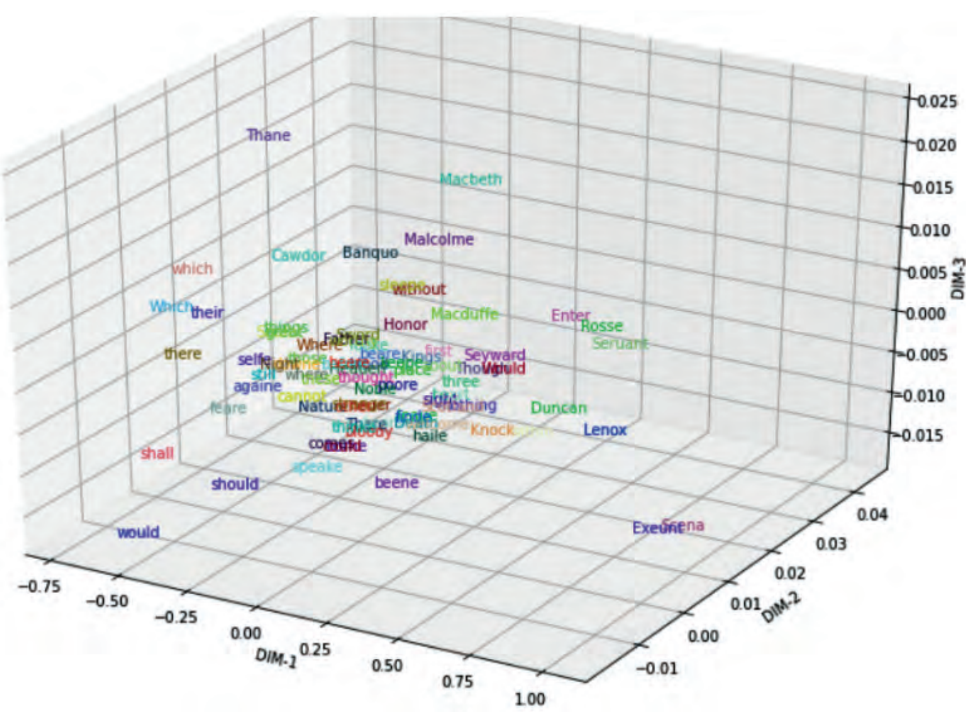

Рис. 1.4. Визуализация векторных представлений слов с помощью PCA

Модели, подобные Word2vec, изучают представление слов с помощью нейронной архитектуры, основанной на прогнозировании. Они используют градиентный спуск для некоторых целевых функций и предсказаний ближайших слов. В то время как традиционные подходы основаны на подсчете, архитектуры нейронных моделей основаны на прогнозировании дистрибутивной семантики. Какие методы лучше подходят для дистрибутивных представлений слов – основанные на подсчете или основанные на прогнозировании? Разработчики метода GloVe отвергают такую постановку вопроса, утверждая, что эти два подхода кардинально не отличаются. Джеффри Пенингтон и некоторые другие исследователи даже склоняются к идее о том, что методы, основанные на подсчете, могут быть более успешными, если собирать глобальную статистику. Они заявили, что метод GloVe превзошел другие языковые модели нейронных сетей в задачах аналогии слов, сходства слов и *распознавания именованных объектов* (named entity recognition, NER).

Однако эти два подхода не предлагают нам полезные решения проблем невидимых слов и неоднозначности смыслов слов. Они не используют информацию, содержащуюся в частях слов, и поэтому не могут сформировать представления редких и невидимых слов.

В еще одной широко используемой модели FastText предложен новый обогащенный подход с использованием информации, содержащейся в частях слов, где каждое слово представлено в виде набора *n*-грамм символов. Модель

задает постоянный вектор для каждой *n*-граммы символа и представляет слова как суммы векторов их частей. Эту идею впервые предложил Хинрих Шютце (Hinrich Schütze; *Word Space*, 1993). Модель может вычислять представления даже для невидимых слов и изучать внутреннюю структуру слов, такую как суффиксы/приставки, что особенно важно для морфологически богатых языков, таких как финский, венгерский, турецкий, монгольский, корейский, японский, индонезийский и др. В настоящее время современные трансформерные архитектуры используют различные методы токенизации частей слов, такие как WordPiece, SentencePiece или Byte-Pair Encoding (BPE).

Краткий обзор RNN

Модели на основе рекуррентной нейросети могут изучать представление каждого токена путем объединения информации о других токенах на более раннем временном шаге и изучения представления предложения на текущем временном шаге. Оказалось, что этот механизм хорошо работает во многих сценариях:

- во-первых, RNN может быть преобразована в модель «один ко многим» для генерации текстов или музыки;
- во-вторых, модели «многие к одному» можно использовать для классификации текста или анализа тональности;
- и наконец, для задач NER используются модели «многие ко многим». Другое применение моделей «многие ко многим» – решение задач кодировщика-декодера, таких как машинный перевод, ответы на вопросы и резюмирование текста.

Как и в случае с другими нейросетевыми моделями, модели RNN принимают токены, созданные с помощью алгоритма токенизации, разбивающего весь исходный текст на элементарные единицы, также называемые токенами. Кроме того, он связывает отдельные токены с числовыми векторами – представлениями токенов, которые изучаются во время обучения. В качестве альтернативы мы можем заблаговременно назначить задачу изучения представлений известным алгоритмам представления слов, таким как Word2vec или FastText.

Рассмотрим простой пример архитектуры RNN для предложения *The cat is sad.* (Кошка грустна), где x_0 – векторное представление слова *the*, x_1 – векторное представление слова *cat* и т. д. На рис. 1.5 показано условное (развернутое) представление RNN в виде *глубокой нейронной сети* (deep neural network, DNN).

Развертывание RNN (unfolding, иногда говорят *раскрытие*) в данном случае означает, что мы связываем с каждым словом отдельный слой нейросети. Получив предложение *The cat is sad.*, мы должны обработать последовательность из пяти слов (включая точку). Скрытое состояние на каждом слое действует как память сети. Она кодирует информацию о том, что произошло во всех предыдущих временных шагах и в текущем временном шаге, как показано на рис. 1.5.

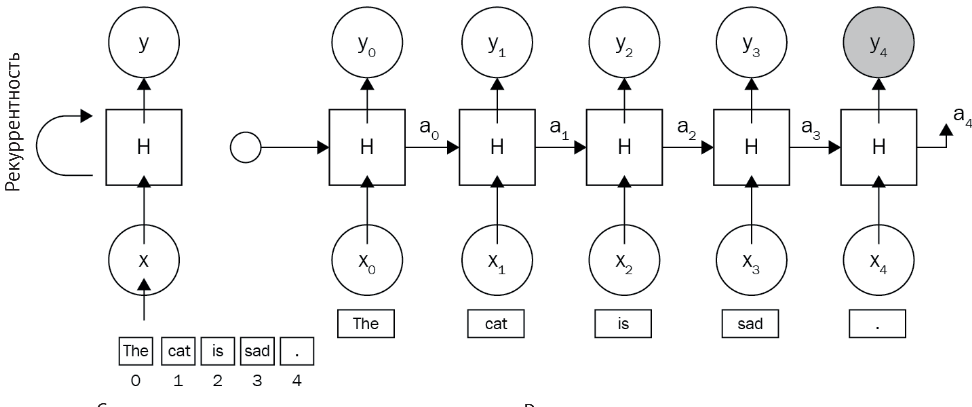

Рис. 1.5. Архитектура RNN

Архитектура RNN обладает следующими преимуществами:

- **входные данные переменной длины:** возможность работать с вводом переменной длины независимо от размера вводимого предложения. Мы можем вводить в сеть предложения из 3 или 300 слов без изменения параметров;
- **учитывает порядок слов:** RNN обрабатывает последовательность слово за словом по порядку, обращая внимание на положение слов;
- **подходит для работы в различных режимах (многие ко многим, один ко многим):** мы можем обучить модель выполнять машинный перевод или анализировать тональность, используя общую идею рекуррентности. Обе архитектуры будут основаны на RNN.

В то же время архитектура RNN обладает и недостатками:

- **проблема отдаленной зависимости:** когда мы обрабатываем очень длинный документ и пытаемся связать термины, которые находятся далеко друг от друга, нам нужно принять во внимание и закодировать все другие нерелевантные термины между этими терминами;
- **склонность к проблемам с разрывающимся или исчезающим градиентом:** при работе с длинными документами обновление весов самых первых слов является большой проблемой из-за исчезающего градиента, что делает модель необучаемой;
- **трудно применить параллельное обучение:** распараллеливание разбивает основную проблему на более мелкие и выполняет решения одновременно, но RNN основана на классическом последовательном подходе. Состояние каждого слоя сильно зависит от предыдущих слоев, что исключает распараллеливание;
- **медленные вычисления из-за большой длины последовательности:** RNN может быть очень эффективной при работе с коротким текстом. Но более длинные документы она обрабатывает очень медленно (вспомним о проблеме отдаленной зависимости).

Хотя RNN теоретически может извлекать информацию из этапов, задолго предшествующих текущему, в реальном мире такие проблемы, как длинные документы и проблема отдаленной зависимости, не позволяют в полной мере воспользоваться этой возможностью. Проблемы обработки длинных последовательностей в многослойных сетях были предметом многих исследований, например следующих:

- Hochreiter and Schmidhuber. *Long Short-term Memory.* 1997;
- Bengio et al. *Learning long-term dependencies with gradient descent is difficult.* 1993;
- K. Cho et al. *Learning phrase representations using RNN encoder-decoder for statistical machine translation.* 2014.

LSTM и управляемые рекуррентные блоки

Модели LSTM (Schmidhuber, 1997) и GRU (gated recurrent unit; Cho, 2014) – это новые варианты RNN, в которых решена проблема отдаленной зависимости. В частности, различные варианты модели LSTM были специально разработаны для решения проблемы отдаленной зависимости. Преимущество модели LSTM заключается в том, что она использует дополнительное состояние ячейки, которое показано в виде горизонтальной линии в верхней части блока LSTM на рис. 1.6. Это состояние ячейки контролируется специальными гейтами (gate), которые позволяют выполнять операции забывания, вставки или обновления.

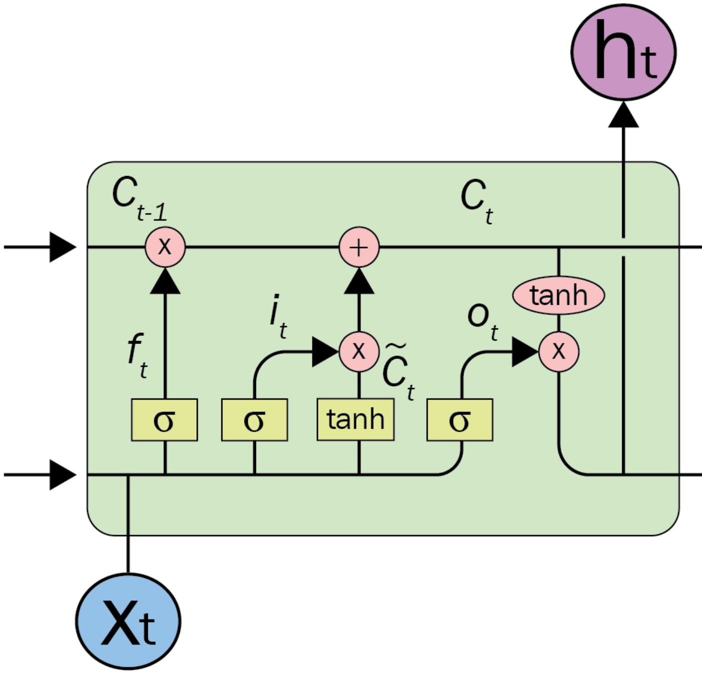

Рис. 1.6. Ячейка LSTM

Мы можем принимать следующие решения:

- какую информацию мы будем хранить в состоянии ячейки;
- какая информация будет забыта (удалена).

Чтобы узнать состояние *I* токенов традиционной RNN, мы вынуждены периодически обрабатывать все состояния предыдущих токенов между $timestep_0$ и $timestep_{i-1}$. Перенос всей информации из более ранних временных шагов порождает проблему исчезающего градиента, что делает модель необучаемой. Механизм гейтов в LSTM позволяет архитектуре пропускать некоторые несвязанные токены на определенном временном шаге или запоминать состояния отдаленных этапов, чтобы узнать текущее состояние токена.

Архитектура модели GRU во многом похожа на LSTM; главное отличие состоит в том, что GRU не использует состояние ячейки. Ее архитектура упрощена за счет переноса функциональности состояния ячейки в скрытое состояние, и она содержит только два гейта – *гейт обновления* и *гейт сброса*. Гейт обновления определяет, сколько информации из предыдущего и текущего временных шагов будет продвинуто вперед. Эта функция помогает модели сохранять актуальную информацию из прошлого, что также сводит к минимуму риск исчезновения градиента. Гейт сброса обнаруживает нерелевантные данные и заставляет модель их забыть.

Мягкая реализация LSTM с помощью Keras

Нам нужно загрузить набор размеченных данных тональности Stanford Sentiment Treebank (SST-2) из теста General Language Understanding Evaluation (GLUE). Мы можем сделать это, выполнив следующие команды в окне терминала:

```
$ wget https://dl.fbaipublicfiles.com/glue/data/SST-2.zip
$ unzip SST-2.zip
```

> **Важное примечание**
>
> SST-2 – это полностью размеченное дерево синтаксического анализа, которое позволяет проводить полный анализ тональности текстов на английском языке. Корпус изначально состоит из 12 тысяч отдельных предложений, взятых из обзоров фильмов. Он был проанализирован с помощью синтаксического анализатора Стэнфорда и включает более 200 тысяч уникальных фраз, каждая из которых аннотирована тремя людьми-экспертами. За дополнительной информацией обратитесь к публикации Socher et al., Parsing With Compositional Vector Grammars, EMNLP. 2013 г. (https://nlp.stanford.edu/sentiment).

После загрузки данных прочитаем их как объект pandas, как показано ниже:

```
import tensorflow as tf
import pandas as pd
df=pd.read_csv('SST-2/train.tsv',sep="\t")
sentences=df.sentence
labels=df.label
```

Нам нужно установить максимальную длину предложения, создать вокабуляр и словари (word2idx, idx2words) и, наконец, представить каждое предложение в виде списка индексов, а не строк. Мы сделаем это, запустив следующий код:

```
max_sen_len=max([len(s.split()) for s in sentences])
words = ["PAD"]+\
            list(set([w for s in sentences for w in s.split()]))
word2idx= {w:i for i,w in enumerate(words)}
max_words=max(word2idx.values())+1
idx2word= {i:w for i,w in enumerate(words)}
train=[list(map(lambda x:word2idx[x], s.split()))\
                          for s in sentences]
```

Последовательности, которые короче max_sen_len (максимальная длина предложения), дополняются значением PAD до тех пор, пока их длина не достигнет max_sen_len. И наоборот, более длинные последовательности усекаются до длины max_sen_len. Эту работу выполняет следующий фрагмент кода:

```
from keras import preprocessing
train_pad = preprocessing.sequence.pad_sequences(train,
                                      maxlen=max_sen_len)
print('Train shape:', train_pad.shape)
Output: Train shape: (67349, 52)
```

Теперь мы готовы разработать и обучить модель LSTM следующим образом:

```
from keras.layers import LSTM, Embedding, Dense
from keras.models import Sequential
model = Sequential()
model.add(Embedding(max_words, 32))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',loss='binary_crossentropy',
metrics=['acc'])
history = model.fit(train_pad,labels, epochs=30, batch_size=32,
validation_split=0.2)
```

Модель будет обучена за 30 эпох. Чтобы получить графическое представление процесса обучения модели LSTM, мы можем выполнить следующий код:

```
import matplotlib.pyplot as plt
def plot_graphs(history, string):
...
plot_graphs(history, 'acc')
plot_graphs(history, 'loss')
```

Этот код создает график, отражающий эффективность обучения и результаты проверки классификации текстов на основе LSTM:

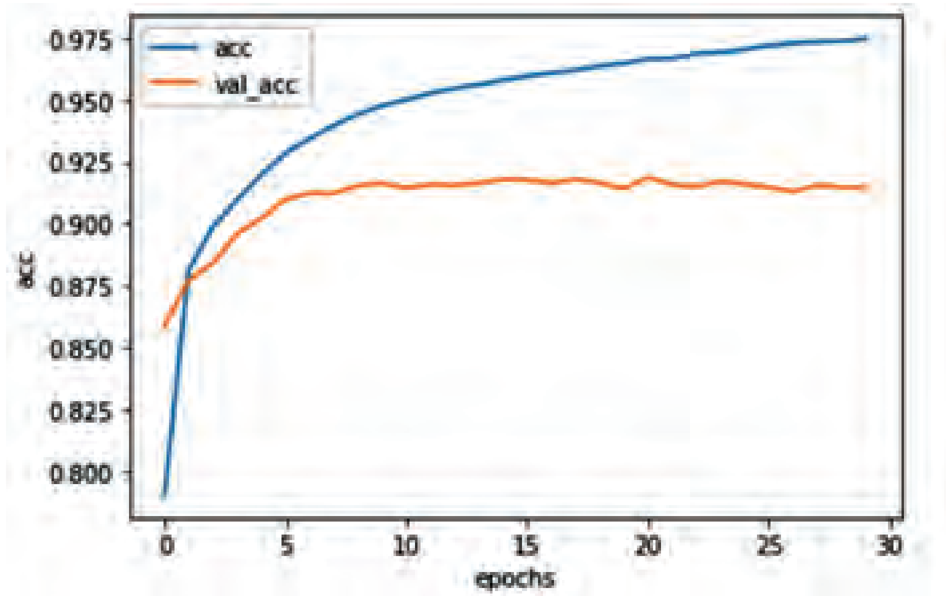

Рис. 1.7. Рост качества классификации сети LSTM по мере обучения

Как мы уже отмечали ранее, основная проблема модели кодировщика-декодера на основе RNN заключается в том, что она создает единственное фиксированное представление текстовой последовательности. Однако механизм внимания позволил RNN сосредоточиться на определенных частях входных токенов, поскольку он сопоставляет их с определенной частью выходных токенов. Этот механизм внимания оказался настолько полезным, что стал одной из основных идей архитектуры трансформеров. Начиная со следующей части этой главы и на протяжении оставшейся книги мы будем говорить о том, как механизм внимания способствует эффективности трансформеров.

Краткий обзор CNN

После того как CNN хорошо зарекомендовали себя в области компьютерного зрения, их начали применять в NLP для моделирования предложений и решения таких задач, как семантическая классификация текста. CNN состоит из сверточных слоев, за которыми во многих случаях следует плотная нейронная сеть. Сверточный слой работает с данными, извлекая полезные признаки. Как и в любой модели глубокого обучения, сверточный слой фактически предназначен для автоматизации извлечения признаков. В случае NLP на этот слой признаков поступают данные от слоя представления, принимающего в качестве входных данных предложения в векторизованном унитарном формате. Унитарные векторы генерируются по `token-id` для каждого слова, образующего предложение. На рис. 1.8 показано представление предложения *I saw a cat.* (*Я видел кошку*) в виде унитарных векторов слов.

	1	2	3	4	5
I	**1**	0	0	0	0
saw	0	**1**	0	0	0
a	0	0	**1**	0	0
cat	0	0	0	**1**	0
.	0	0	0	0	**1**

Рис. 1.8. Представление предложения в виде унитарных векторов

Каждый токен, представленный унитарным вектором, подается на *слой представления* (embedding layer). Слой представления может быть инициализирован случайными значениями или с помощью предварительно обученных векторов слов, таких как GloVe, Word2vec или FastText. Затем предложение будет преобразовано в плотную матрицу с размерностью $N \times E$ (где N – количество токенов в предложении, а E – размер представления). На рис. 1.9 показано, как одномерная CNN обрабатывает эту плотную матрицу.

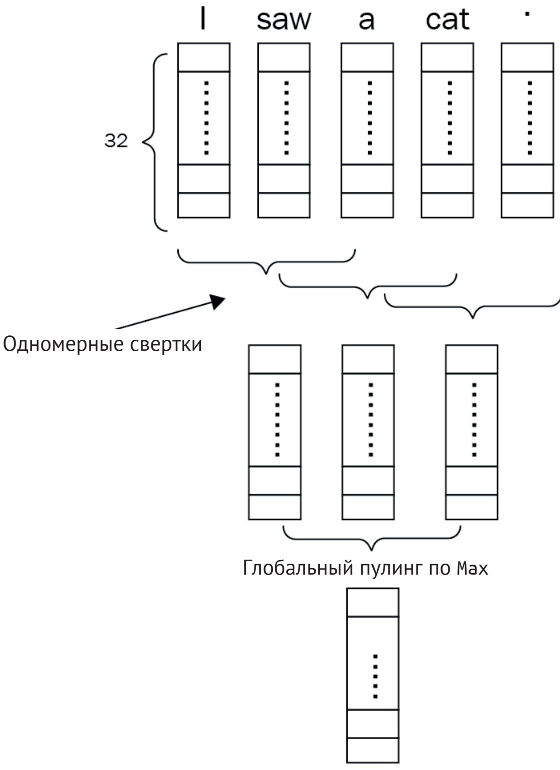

Рис. 1.9. Обработка предложения из пяти токенов в одномерной CNN

Свертка будет самой первой операцией, выполняемой с разными слоями и ядрами. Гиперпараметры сверточного слоя – это размер ядра и количество ядер. Также полезно отметить, что здесь применяется одномерная свертка, и причина этого в том, что представление токенов можно рассматривать только целиком, и мы хотим применить ядра, способные одновременно видеть несколько токенов в последовательном порядке. Вы можете считать это чем-то вроде *n*-граммы с заданным окном. Один из интересных вариантов – использование неглубоких TL в сочетании с моделями CNN. Как показано на рис. 1.10, мы также можем проходить сети с помощью комбинации нескольких представлений токенов, как было предложено в исследовании Yoon Kim, *Convolutional Neural Networks for Sentence Classification* (2014).

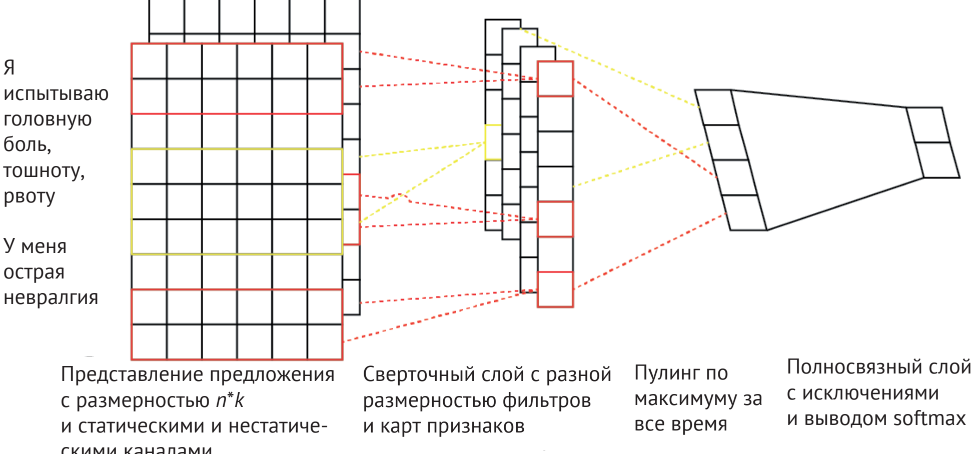

Рис. 1.10. Комбинация нескольких представлений в CNN

Например, мы можем использовать три слоя представления вместо одного и объединить их для каждого токена. При такой схеме определенный токен, например *чувствую*, будет представлен вектором с размерностью 3×128, если размерность всех трех различных представлений равна 128. Эти векторные представления могут быть инициализированы предварительно обученными векторами из Word2vec, GloVe и FastText. Операция свертки на каждом шаге будет видеть *N* слов с соответствующими тремя векторами (*N* – размер фильтра свертки). Тип свертки, который используется здесь, – это одномерная свертка. Размерность – количество направлений перемещения при выполнении операции. Например, двумерная свертка будет перемещаться по двум осям, а одномерная свертка – только по одной оси. На рис. 1.11 показаны различия между ними.

Размерность свертки	Вход	Фильтр	Выход
1-мерная	3-мерный	3-мерный	2-мерный
2-мерная	4-мерный	4-мерный	3-мерный
3-мерная	5-мерный	5-мерный	4-мерный

Рис. 1.11. Направления перемещения свертки

Следующий фрагмент кода представляет собой реализацию одномерной CNN, обрабатывающую те же данные, которые мы использовали в примере с LSTM. Он включает в себя композицию слоев `Conv1D` и `MaxPooling`, за которыми следуют слои `GlobalMaxPooling`. Мы можем настроить параметры и добавить больше слоев для оптимизации модели:

```
from keras import layers
model = Sequential()
model.add(layers.Embedding(max_words, 32, input_length=max_sen_
len))
model.add(layers.Conv1D(32, 8, activation='relu'))
model.add(layers.MaxPooling1D(4))
model.add(layers.Conv1D(32, 3, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1, activation= 'sigmoid')
model.compile(loss='binary_crossentropy', metrics=['acc'])
history = model.fit(train_pad,labels, epochs=15, batch_size=32,
validation_split=0.2)
```

Оказывается, модель CNN демонстрирует производительность, сопоставимую со своим аналогом LSTM. Хотя CNN стали стандартом в обработке изображений, мы видели много успешных применений CNN в области NLP. В то время как модель LSTM обучена распознавать закономерности, развернутые во времени, модель CNN распознает закономерности в пространстве.

Обзор архитектуры трансформеров

Модели-трансформеры вызвали огромный интерес из-за их эффективности в огромном диапазоне задач NLP, от классификации до генерации текста. Критически важной частью этих моделей является механизм внимания. Впрочем, еще до появления трансформеров механизм внимания начали применять для улучшения традиционных моделей глубокого обучения, таких как RNN. Чтобы хорошо разобраться в трансформерах и понять их значение для NLP, мы сначала изучим механизм внимания.

Механизм внимания

Один из первых вариантов механизма внимания был предложен Дмитрием Богдановым[2] в 2015 г. Этот механизм основан на том факте, что модели на основе RNN, такие как GRU или LSTM, имеют информационное узкое место для таких задач, как *нейронный машинный перевод* (neural machine translation, NMT). Эти модели на основе энкодера-декодера получают ввод в виде `token-id` и подвергают его рекуррентной обработке (энкодер). После этого обработанное промежуточное представление подается в другой рекуррентный блок (декодер) для извлечения результатов. Эта лавинообразная информация подобна катящемуся снежному шару, на который налипает вся информация, и «раскатать» этот шар обратно – крайне сложная задача, потому что декодирующая часть не видит всех зависимостей и получает в качестве входных данных только промежуточное представление (вектор контекста).

Чтобы устранить это узкое место и согласовать декодер с энкодером, Богданов предложил механизм внимания, в котором используются веса, присвоенные промежуточным скрытым значениям. Эти веса определяют количество внимания, которое модель должна уделять входным данным на каждом этапе декодирования. Наличие подобных замечательных подсказок очень помогает моделям в таких задачах, как NMT, которые относятся к задачам типа «многие ко многим». На рис. 1.12 представлена схема типичного механизма внимания.

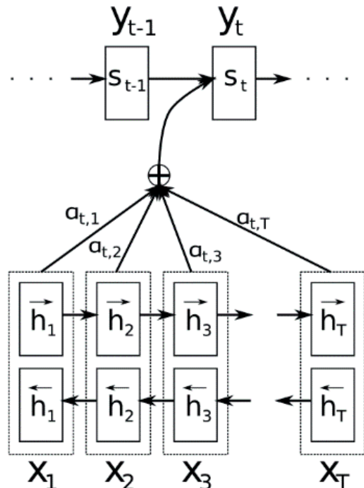

Рис. 1.12. Схема типичного механизма внимания

Впоследствии были предложены разные механизмы внимания с разнообразными улучшениями. В семейство этих механизмов входят *аддитивное*, *мультипликативное*, *общее* внимание, а также *внимание на основе скалярного*

[2] Dzmitry Bahdanau (http://rizar.github.io/) – канадский исследователь и разработчик в области NLP и ИИ белорусского происхождения, работает в университете Макгилла в Монреале, Канада.

произведения (dot-product attention, DPA)[3]. Модифицированная версия последнего механизма, имеющая весовой параметр, известна как *взвешенное скалярное произведение* (scaled dot-product). Этот особый тип внимания лежит в основе моделей трансформеров. Мы будем с определенной долей условности называть его *механизмом многопоточного внимания* (multi-head attention mechanism)[4]. Механизм аддитивного внимания также оказал заметное влияние на задачи NMT. В табл. 2 представлен обзор различных типов механизмов внимания.

Таблица 2. Типы механизмов внимания (на основе https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html)

Название	Функция внимания	Источник
Контентное внимание (content-based attention)	$\mathrm{score}(\boldsymbol{s}_i, \boldsymbol{h}_i) = \cos[\boldsymbol{s}_i, \boldsymbol{h}_i]$	Грейвс (Graves, 2004)
Аддитивное (additive)	$\mathrm{score}(\boldsymbol{s}_i, \boldsymbol{h}_i) = \mathbf{v}_a^{\top}\tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Богданов (Bahdanau, 2015)
Позиционное (location-based)	$\alpha_{t,i} = \mathrm{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$	Луонг (Luong, 2015)
Общее (general)	$\mathrm{score}(\boldsymbol{s}_i, \boldsymbol{h}_i) = s_t^{\top}\mathbf{W}_a \boldsymbol{h}_i$	Луонг (Luong, 2015)
Скалярное произведение (dot-product)	$\mathrm{score}(\boldsymbol{s}_i, \boldsymbol{h}_i) = s_t^{\top}\boldsymbol{h}_i$	Луонг (Luong, 2015)
Взвешенное скалярное произведение (scaled dot product)	$\mathrm{score}(\boldsymbol{s}_i, \boldsymbol{h}_i) = \frac{s_t^{\top}\boldsymbol{h}_i}{\sqrt{n}}$	Васвани (Vaswani, 2017)

Поскольку механизмы внимания не являются специфическими для NLP, их также применяют в различных областях и сценариях, от компьютерного зрения до распознавания речи. На рис. 1.13 показана визуализация процесса мультимодального обучения нейросети созданию подписей к изображениям (K Xu et al., *Show, attend and tell: Neural image caption generation with visual attention*, 2015).

[3] Обычно термин dot-product attention не переводят на русский язык, поэтому дальше мы будем использовать аббревиатуру DPA. – *Прим. перев.*

[4] Этот термин тоже обычно не переводят на русский язык либо дают кальку «многоголовое внимание». Но, поскольку суть этого подхода заключается либо в многократном повторении операции взвешенного скалярного произведения в одном потоке, либо в многопоточном вычислении, мы предлагаем перевод «многопоточное внимание», который будем использовать в этой книге. Определенная игра слов заключается в том, что обычный вычислительный поток называется thread, а в контексте механизма внимания поток называют head. – *Прим. перев.*

Подпись: Женщина бросает фрисби в парке

Рис. 1.13. Механизм внимания в компьютерном зрении

Механизм многопоточного внимания, схематически представленный на рис. 1.14, является важной частью архитектуры трансформеров:

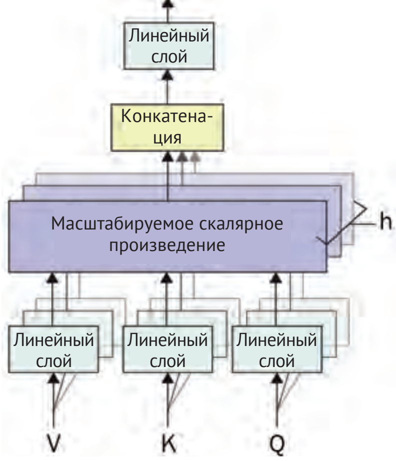

Рис. 1.14. Механизм многопоточного внимания

Давайте рассмотрим механизмы многопоточного внимания более детально.

Механизмы многопоточного внимания

Прежде чем переходить к механизмам внимания на основе взвешенных скалярных произведений, необходимо разобраться в том, что такое *самовнимание* (self-attention). Механизм самовнимания, схематически представленный на рис. 1.15, является базовой формой механизма *взвешенного самовнимания*

(scaled self-attention). Этот механизм получает матрицу ввода *X* и производит оценку внимания ее элементов. В нашем случае *X* представляет собой матрицу 3×4, где 3 – количество токенов, а 4 – размер представления. Матрица *Q* на рис. 1.15 называется *запросом* (query), *K* – *ключом* (key), а *V* – *значением* (value). Эти матрицы получаются в результате перемножения *X* и матриц, помеченных символами *Θ*, *Φ* и *g* соответственно. Перемножение матриц запроса (*Q*) и ключа (*K*) дает матрицу весов внимания. Этот механизм также можно рассматривать как базу данных, в которой мы используем запрос и ключи, чтобы получить численную оценку связей между различными элементами. Перемножение весов внимания и матрицы *V* дает конечный результат механизма внимания этого типа. Основная причина, по которой он называется *самовниманием*, – это его единственный вход *X*; матрицы *Q*, *K* и *V* вычисляются из *X*.

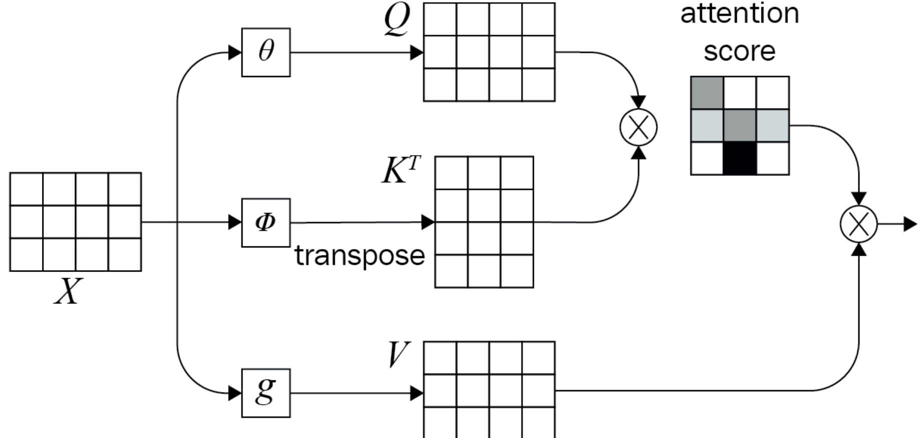

Рис. 1.15. Математическое представление механизма внимания (на основе схемы с сайта https://blogs.oracle.com/datascience/multi-head-self-attention-in-nlp)

Механизм взвешенного DPA очень похож на механизм самовнимания (скалярного произведения), за исключением того, что он использует весовой коэффициент. С другой стороны, многопоточность гарантирует, что модель способна рассматривать различные аспекты ввода на всех уровнях. Модели-трансформеры учитывают аннотации энкодера и скрытые значения из прошлых слоев. Архитектура трансформера не имеет рекуррентного пошагового потока; вместо этого в ней применяется позиционное кодирование, чтобы иметь информацию о позиции каждого токена во входной последовательности. На слои в первой части энкодера подаются входные данные, представляющие собой объединенные значения представлений (инициализированные случайным образом) и фиксированные значения позиционного кодирования; они распространяются по архитектуре, как показано на рис. 1.16.

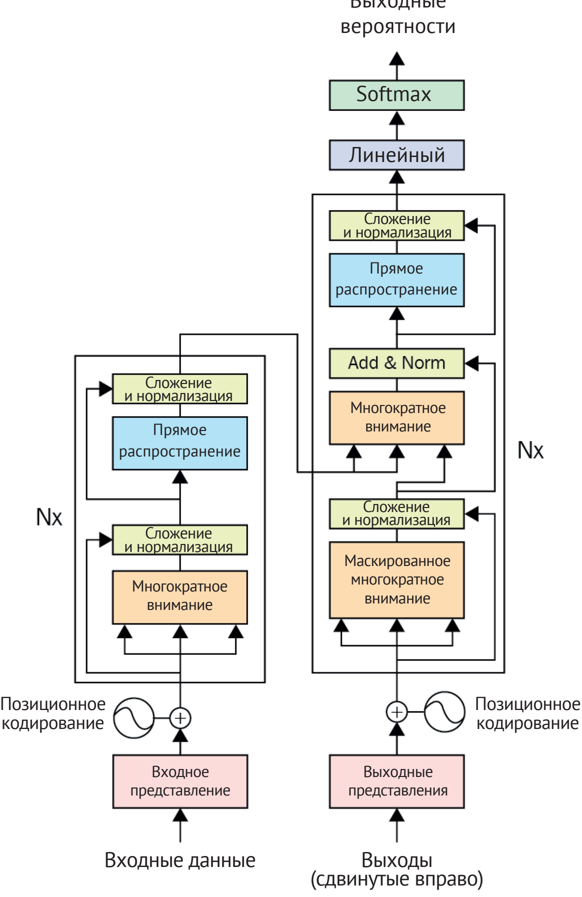

Рис. 1.16. Обобщенная схема трансформера

Информация о положении получается путем вычисления синусоидальных и косинусоидальных волн на разных частотах. Пример позиционного кодирования представлен на рис. 1.17.

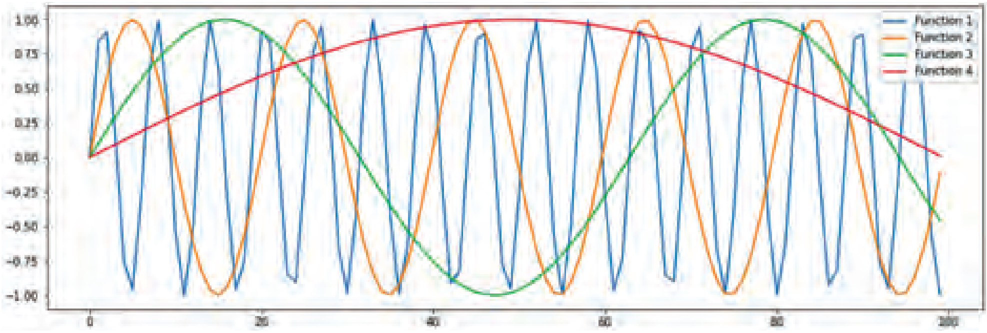

Рис. 1.17. Позиционное кодирование (на основе изображения с сайта https://jalammar.github.io/illustrated-transformer/)

Хороший пример качества работы трансформера и механизма взвешенного DPA приведен на широко известной схеме (рис. 1.18), которая встречается во многих публикациях.

Рис. 1.18. Пример работы механизма внимания трансформера (на основе изображения с сайта https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html)

Как видно на рис. 1.18, слово *it* в разных контекстах обозначает разные объекты. (Например, в предложении ***Курица*** *не перешла дорогу, потому что* ***она*** *была слишком испугана* слово *она* связано со словом *курица*. В предложении *Курица не перешла* ***дорогу***, *потому что* ***она*** *была слишком широкая* слово *она* связано со словом *дорога*). Еще одно улучшение, которое стало возможным благодаря трансформерам, – это параллелизм. Обычные последовательные рекуррентные модели, такие как LSTM и GRU, не имеют подобных возможностей, потому что они обрабатывают входные данные токен за токеном. С другой стороны, слои прямого распространения ускоряются немного больше, потому что умножение одной матрицы происходит значительно быстрее, чем рекуррентная операция. Стеки слоев многопоточного внимания позволяют лучше понимать сложные предложения. Хороший наглядный пример механизма многопоточного внимания представлен на рис. 1.19.

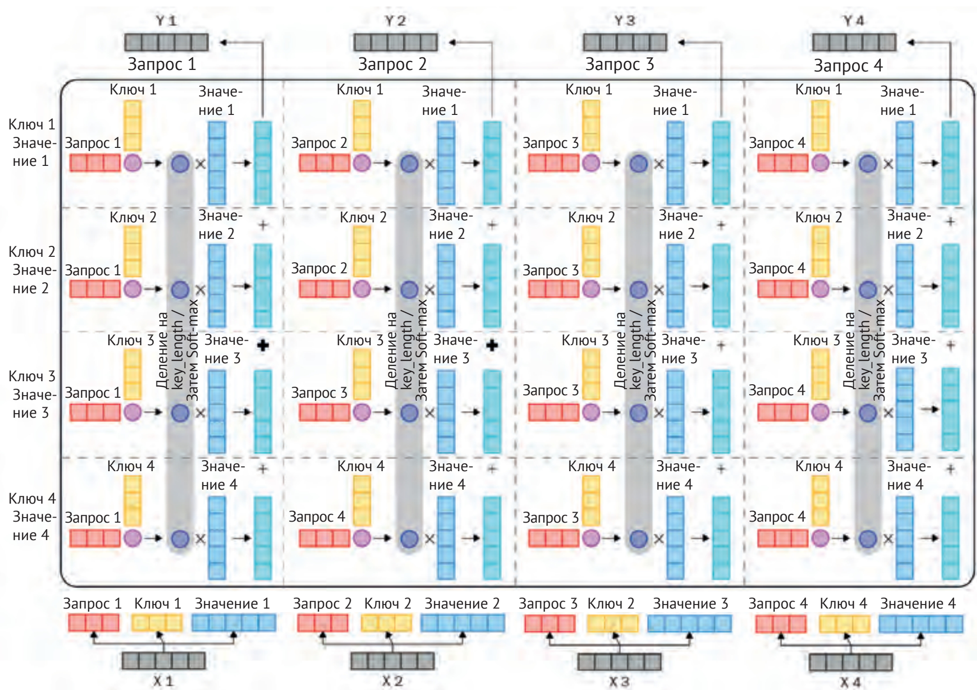

Рис. 1.19. Механизм многопоточного внимания
(на основе изображения с сайта https://imgur.com/gallery/FBQqrxw)

На стороне декодера механизма внимания используется подход, очень похожий на энкодер, но с небольшими изменениями. Механизм многопоточного внимания устроен аналогично, но в нем также используется выход стека энкодера, откуда данные передаются каждому стеку декодера на втором уровне механизма многопоточного внимания. Эта небольшая модификация открывает модели доступ к стеку энкодера в процессе декодирования и в то же время помогает ей во время обучения поддерживать лучший градиент потока через слои. Последний слой Softmax за слоем декодера предоставляет выходные данные для различных задач, таких как NMT (для которых и была разработана оригинальная архитектура трансформеров).

Эта архитектура имеет два канала, обозначенных как входы и выходы (сдвинутые вправо). Один из них всегда присутствует (входы) как при обучении, так и при выводе, в то время как другой присутствует только при обучении и выводе, который производится моделью. Причина, по которой мы не используем прогнозы модели для вывода, заключается в том, чтобы не позволить модели зайти слишком далеко по неправильному пути. Но что это значит? Представьте себе нейронную модель перевода, которая пытается перевести предложение с английского на французский – на каждом шаге она предсказывает слово и использует его для предсказания следующего слова. Но если на каком-то этапе что-то пойдет не так, все последующие прогнозы тоже окажутся неверными. Чтобы модель не пошла по неправильному пути, мы предоставляем правильные слова в виде версии со сдвигом вправо.

Наглядный пример модели трансформера приведен на рис. 1.20. Здесь показана модель трансформера с двумя энкодерами и двумя слоями декодера. Слой «Сложение и нормализация» на этой схеме прибавляет и нормализует ввод, который он принимает от слоя «Прямое распространение»:

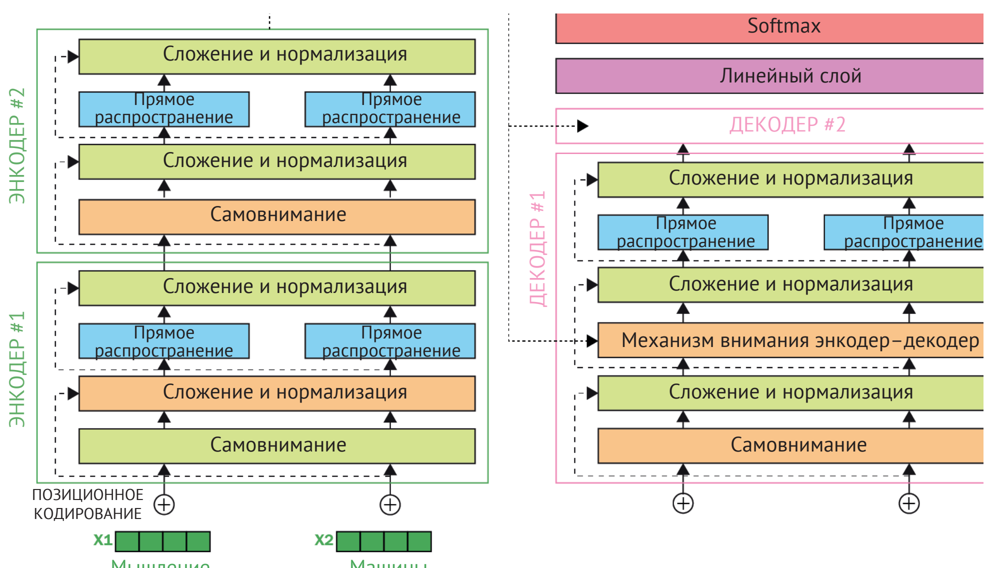

Рис. 1.20. Модель трансформера
(на основе изображения с сайта https://jalammar.github.io/illustrated-transformer/)

Другое важное усовершенствование, которое используется в архитектуре на основе трансформеров, основано на простой универсальной схеме сжатия текста для предотвращения появления невидимых токенов на стороне ввода. Этот подход, который осуществляется с использованием различных методов, таких как кодирование пар байтов и кодирование частей предложения, улучшает качество работы трансформеров с невидимыми токенами. Он также помогает модели, когда ей встречаются морфологически близкие токены. Такие токены раньше никогда не попадались и редко применялись при обучении, но тем не менее их можно использовать для вывода. В некоторых случаях при обучении встречаются их фрагменты; последнее происходит в случае морфологически богатых языков, таких как турецкий, немецкий, чешский и латышский. Например, модель знает слово *домик*, но не *домики*. В таких случаях она может токенизировать слово *домики* как *домик+и*. Обе эти части модели известны.

Модели на основе трансформеров имеют много общего – например, все они основаны на общей оригинальной архитектуре, и разница лишь в том, какие шаги они используют, а какие не используют. В некоторых случаях вносятся незначительные изменения, например улучшения в механизме многопоточного внимания.

Трансформеры и перенос обучения

Перенос обучения – это область искусственного интеллекта и машинного обучения, цель которой – сделать модели повторно используемыми для различных задач. Например, модель, обученная для определенной задачи *A*, может быть повторно использована (тонко настроена) для другой задачи *B*. В области NLP этого добиваются с помощью таких архитектур, как трансформеры, которые могут сформировать понимание языка как такового, с помощью языкового моделирования. Такие модели называются *языковыми моделями* – они предоставляют модель языка, на котором их обучали. Перенос обучения – это не новый подход, он активно применяется в различных областях, таких как компьютерное зрение. В качестве примеров предварительно обученных моделей можно назвать ResNet, Inception, VGG и EfficientNet, которые можно настраивать для различных задач компьютерного зрения.

В области NLP также возможно *неглубокое* (shallow) обучение с переносом с использованием таких моделей, как Word2vec, GloVe и Doc2vec. Такое обучение называется неглубоким, потому что в основе этого типа обучения нет модели, а вместо нее используются предварительно обученные векторы для слов/токенов. Вы можете использовать эти модели представления токенов или документов, за которыми следует классификатор, или применять их в сочетании с другими моделями, такими как RNN, вместо использования случайных представлений.

Перенос обучения в NLP с использованием трансформеров также возможен, потому что эти модели способны изучать сам язык без каких-либо размеченных данных. Моделирование языка – это задача получения переносимых весов для решения различных задач. *Масочное моделирование языка* (masked language modeling) – это один из методов, используемых для изучения самого языка. Как и в случае с оконной моделью Word2vec для прогнозирования центральных токенов, при масочном моделировании языка применяется аналогичный подход с некоторыми важными отличиями. При заданной вероятности каждое слово маскируется и заменяется специальным токеном, например [MASK]. Языковая модель (в нашем случае модель на основе трансформера) должна предсказывать замаскированные слова. В отличие от Word2vec, где используется окно, на вход модели подается целое предложение, и на выходе должно быть то же предложение с восстановленными замаскированными словами.

Одной из первых моделей, в которых для языкового моделирования использовалась архитектура трансформеров, является BERT, основанная на энкодерной части трансформера. Масочное моделирование языка выполняется BERT с использованием того же метода, который был описан до и после обучения языковой модели. BERT – это переносимая языковая модель для различных задач в области NLP, таких как классификация токенов, классификация последовательностей или даже ответы на вопросы.

Каждая из упомянутых задач относится к задачам тонкой настройки BERT после обучения языковой модели. Модель BERT наиболее известна своими ключевыми характеристиками базовой модели трансформерного энкодера, и, изменяя эти характеристики, можно получить разные версии – малые, облегченные, базовые, большие и сверхбольшие. Контекстное представление по-

зволяет модели извлекать правильное значение каждого слова в зависимости от контекста, в котором оно дано, – например, слово *холодный* может иметь разные значения в двух разных предложениях: *холодный взгляд* и *холодный ветер*. Ключевыми характеристиками модели являются количество слоев энкодерной части, входная и выходная размерности представления и количество механизмов многопоточного внимания, как показано на рис. 1.21.

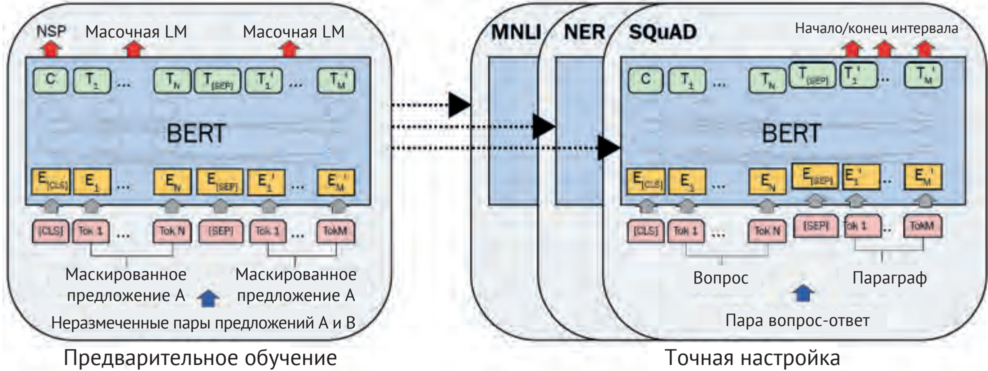

Рис. 1.21. Процедуры предварительного обучения и тонкой настройки BERT (на основе публикации J. Devlin et al., Bert: Pre-training of deep bidirectional Transformers for language understanding, 2018)

Как показано на рис. 1.21, предварительная фаза обучения также содержит другую цель, известную как *прогнозирование следующего предложения* (next-sentence prediction). Как мы знаем, каждый документ состоит из предложений, следующих друг за другом, и еще одна важная часть обучения модели пониманию языка заключается в понимании отношений между предложениями – другими словами, связаны они между собой или нет. Для решения этих задач в BERT имеются специальные токены, такие как [CLS] и [SEP]. Не имеющий изначального смысла токен [CLS] применяется в качестве стартового токена для всех задач и содержит всю информацию о предложении. В задачах классификации последовательности, таких как NSP, используется классификатор поверх выходных данных этого токена (выходная позиция 0). Он также полезен для оценки смысла предложения или получения его семантики, например при использовании сиамской модели BERT очень полезно сравнить два токена [CLS] для разных предложений по такому критерию, как косинусное подобие. В свою очередь, токен [SEP] используется только для различения двух предложений. Если после предварительного обучения кто-то захочет настроить BERT для задачи классификации последовательностей, такой как анализ тональности, ему следует использовать классификатор поверх выходного представления [CLS]. Также примечательно, что все модели с переносом обучения можно «заморозить» при тонкой настройке или, наоборот, «разморозить». Замораживание означает остановку обучения, после чего мы можем видеть все веса и смещения внутри модели как константы. В случае анализа тональности будет обучаться только классификатор, а не модель, если она заморожена.

Заключение

После прочтения этой главы у вас должно возникнуть понимание эволюции методов и подходов NLP от BoW до трансформеров. Вы узнали, как реализовать подходы на основе BoW, RNN и CNN, что такое Word2vec и как он помогает улучшить традиционные методы на основе глубокого обучения с использованием неглубокого переноса. В качестве примера мы рассмотрели основы архитектуры модели BERT. В конце главы обсудили перенос обучения и его применение совместно с BERT.

На этом этапе вы получили основную информацию, которая необходима для перехода к следующим главам. Теперь вы должны понимать основную идею архитектуры трансформеров и представлять, как можно применять перенос обучения вместе с этой архитектурой.

В следующей главе мы представим простой пример создания модели трансформера с нуля. Мы расскажем об этапах установки, а также подробно продемонстрируем работу с наборами данных и эталонными тестами.

Дополнительная литература

- Mikolov, T., Chen, K., Corrado, G. & Dean, J. (2013). *Efficient estimation of word representations in vector space.* arXiv preprint arXiv:1301.3781.
- Bahdanau, D., Cho, K. & Bengio, Y. (2014). *Neural machine translation by jointly learning to align and translate.* arXiv preprint arXiv:1409.0473.
- Pennington, J., Socher, R. & Manning, C. D. (2014, October). *GloVe: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532–1543).
- Hochreiter, S. & Schmidhuber, J. (1997). *Long short-term memory.* Neural computation, 9(8), 1735–1780.
- Bengio, Y., Simard, P, & Frasconi, P. (1994). *Learning long-term dependencies with gradient descent is difficult.* IEEE transactions on neural networks, 5(2), 157–166.
- Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. & Bengio, Y. (2014). *Learning phrase representations using RNN encoder-decoder for statistical machine translation.* arXiv preprint arXiv:1406.1078.
- Kim, Y. (2014). *Convolutional neural networks for sentence classification.* CoRR abs/1408.5882 (2014). arXiv preprint arXiv:1408.5882.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N. & Polosukhin, I. (2017). *Attention is all you need.* arXiv preprint arXiv:1706.03762.
- Devlin, J., Chang, M. W., Lee, K. & Toutanova, K. (2018). *Bert: Pre-training of deep bidirectional Transformers for language understanding.* arXiv preprint arXiv:1810.04805.