20
ВНИМАНИЕ И ТРАНСФОРМАТОРЫ

В главе 19 мы рассмотрели, как использовать RNN для работы с последовательными данными. Несмотря на свою мощь, RNN имеют ряд недостатков. Поскольку вся информация о входных данных представлена в одном фрагменте Память состояний или вектор контекста, сети внутри каждой рекуррентной ячейки приходится напрягаться, чтобы втиснуть все необходимое в доступное пространство. И какой бы объем памяти состояний мы ни сделали, всегда можно получить входной сигнал, превышающий объем памяти, поэтому что-то обязательно теряется.

Другая проблема заключается в том, что RNN должна обучаться и использоваться по одному слову за раз.
Это может быть медленным способом работы, особенно с большими базами данных.

Альтернативный подход основан на использовании небольшой сети, называемой *сетью внимания,* которая не имеет памяти состояний и может обучаться и использоваться параллельно. Сети внимания могут объединяться в более крупные структуры, называемые *трансформаторами,* которые способны выступать в качестве языковых моделей, способных выполнять такие задачи, как перевод. Строительные блоки трансформаторов могут быть использованы в других архитектурах, обеспечивающих еще более мощные языковые модели, в том числе в генераторах.

В этой главе мы начнем с более мощного способа представления слов, а не отдельных чисел, а затем перейдем к вниманию и современным

архитектуры, использующие блоки трансформации для выполнения многих задач НЛП.

Встраивание

В главе 19 мы обещали улучшить наши описания слов, не ограничиваясь одним числом. Ценность этого изменения заключается в том, что оно позволяет нам манипулировать представлениями слов в значимых направлениях. Например, мы можем найти слово, похожее на другое слово, или смешать два слова, чтобы найти то, которое находится между ними. Эта концепция является ключевой для развития внимания, а затем и трансформации.

Эта техника называется *встраиванием слов* (или *встраиванием токенов,* когда мы используем ее для более общего понятия "токен"). Она несколько абстрактна, поэтому давайте рассмотрим ее суть на конкретном примере.

Предположим, что вы работаете дрессировщиком на съемках фильма с буйным режиссером. Сегодня вы снимаете эпизод, в котором героев-людей преследуют какие-то животные. Режиссер просит у вас список животных, которых вы можете предоставить в достаточном количестве, чтобы получилась страшная погоня. Вы звоните в свой офис, они готовят список и даже выстраивают его в виде графика, где по горизонтальной оси откладывается средняя скорость каждого взрослого животного, а по вертикальной - его средний вес, как показано на *рис. 20-1*.

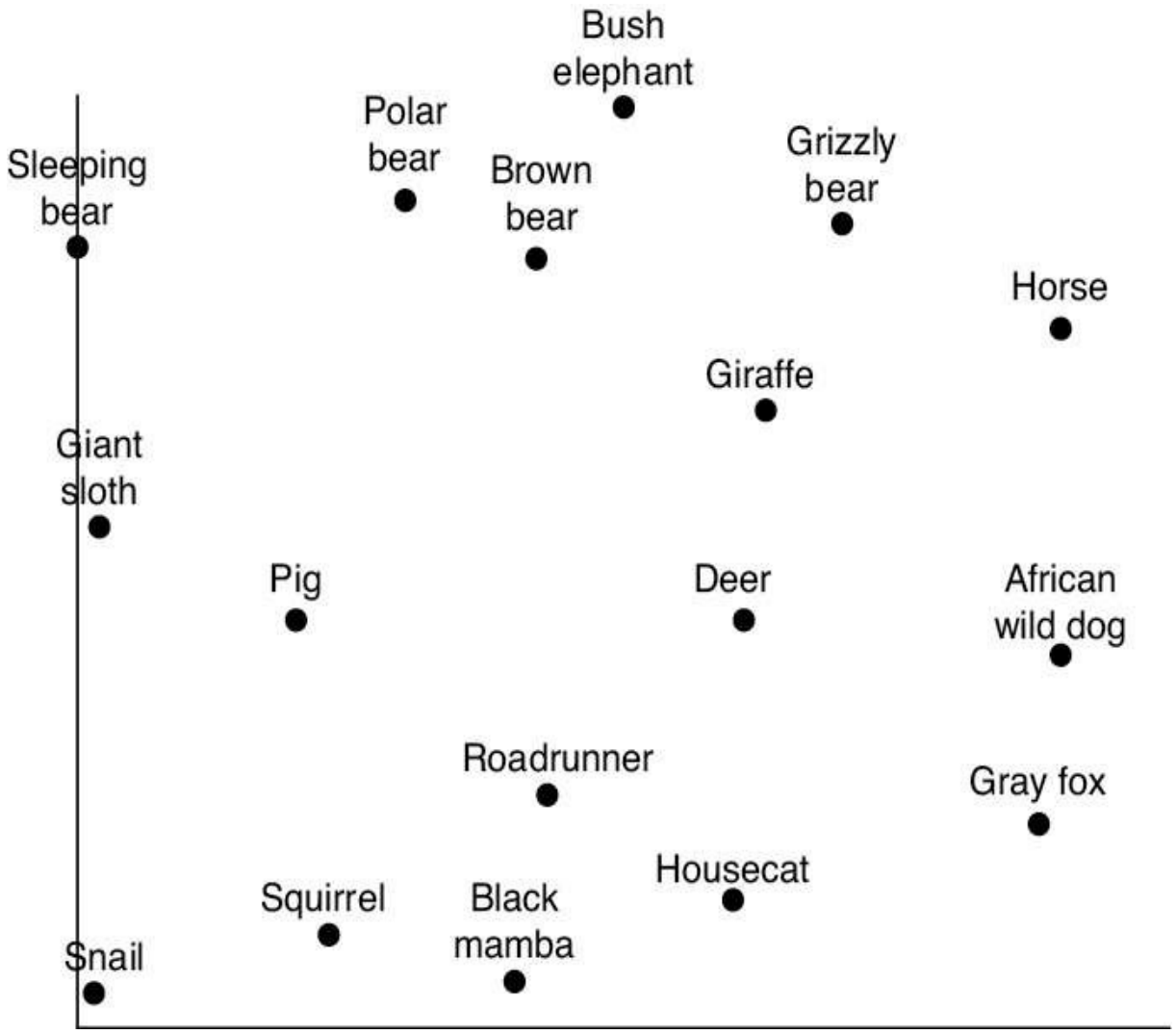

Рис. 20-1: Коллекция животных, упорядоченная по скорости передвижения по горизонтали и по весу взрослого человека по вертикали, хотя обозначения по осям отсутствуют *(данные Reisner 2020)*

Но из-за ошибки принтера на диаграмме, которую прислал вам офис, отсутствуют метки на осях, поэтому у вас есть диаграмма с животными в двухмерном изображении, но вы не знаете, что означают оси.

Директор даже не смотрит на карту. "Лошади, - говорит она, - мне нужны лошади".
Они именно то, что я хочу, и будут идеальны, и ничто другое не подойдет".
Итак,
Вы приводите лошадей, и они репетируют сцену.

К сожалению, директор недоволен. "Нет, нет, нет!" - говорит она. "Лошади слишком дерганые и быстрые. Они похожи на лис. Дайте мне лошадей, которые меньше похожи на лис".

Как вы можете удовлетворить этот запрос? Что это вообще значит?
Счастливо,

Вы можете сделать то, что она просит, с помощью chan, просто объединив стрелки.

Со стрелками можно делать только две вещи: складывать их и вычитать. Чтобы добавить стрелку B к стрелке A, поместите хвост B на головку A. Новая стрелка A
+ B начинается в хвосте A, а заканчивается в голове B, как в середине *рис. 20-2*.

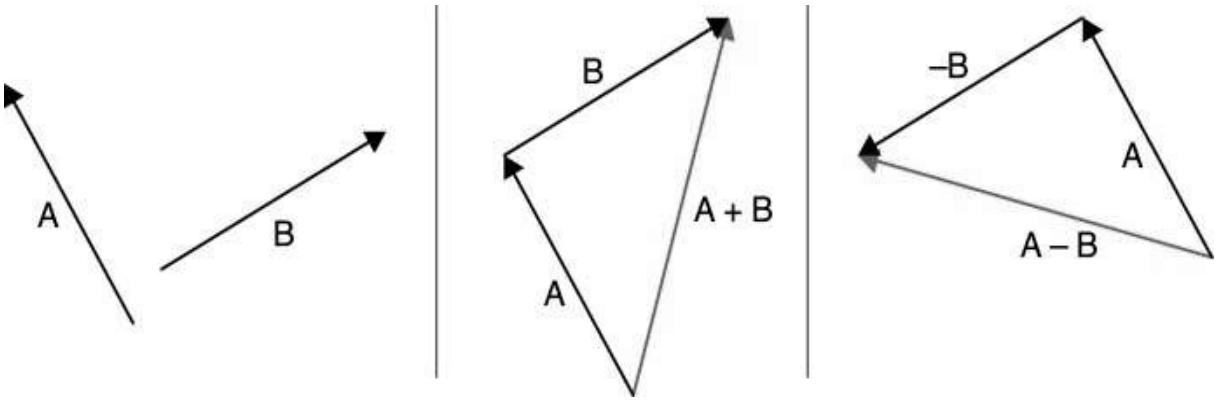

Рисунок 20-2: Стрелочная арифметика. Лефи: 2 стрелки. Средняя: Сумма A + B. Справа: Разность A - B.

Чтобы вычесть B из A, достаточно перевернуть B на 180 градусов, чтобы получилось -B, и сложить A и -B. Результат, A - B, начинается в хвосте A и заканчивается в голове -B, как показано справа на *рис. 20-2*.

Теперь можно удовлетворить желание режиссера убрать лисьи качества у лошадей. Начните с того, что нарисуйте стрелку из левого нижнего угла чань к лошади, а другую - к лисе, как в левой части ***фигуры*** 20-3.

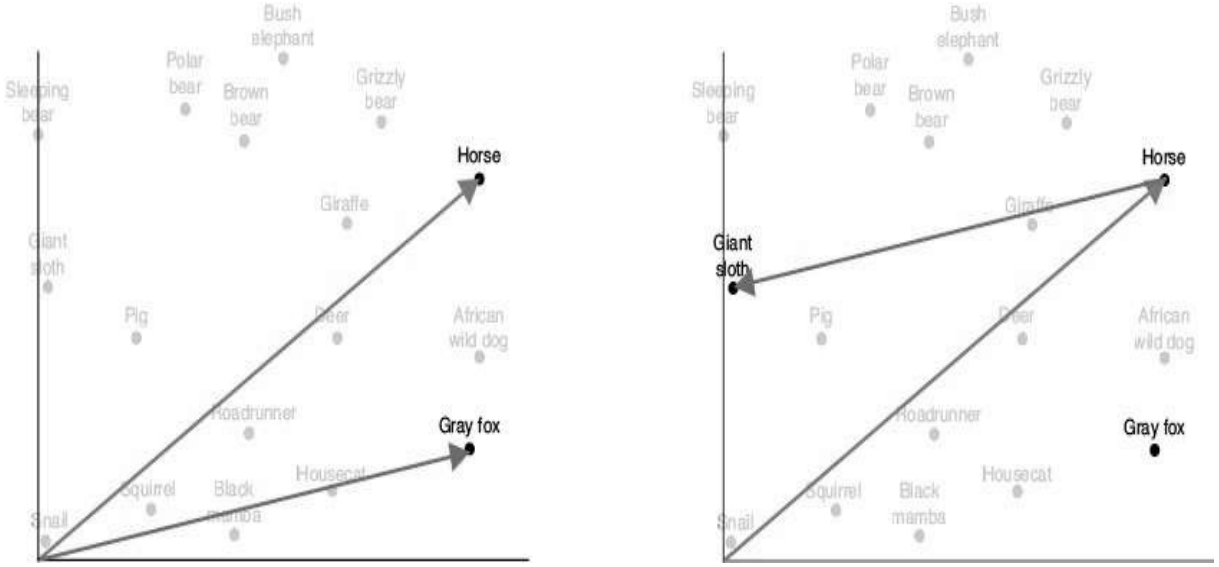

Рисунок 20-3: Слева: стрелки от нижнего лефи к лошади и лисе. Справа: Вычитание лисы из лошади дает нам гигантского ленивца.

Теперь вычтите лисицу из лошади, как было предложено, вычитая стрелу лисицы из стрелы лошади. В соответствии с правилами *рис. 20-2* это означает, что нужно перевернуть стрелку лисы и поместить ее хвост в голову стрелки лошади. Получаем правую часть *рис. 20-3*.

Гигантский ленивец. Ну, хорошо, так захотел режиссер. Можно даже записать это как арифметику: лошадь - лиса = гигантский ленивец (по крайней мере, по нашей схеме).

Директор бросает свой латте на пол. "Нет, нет, нет! Конечно, ленивцы выглядят отлично, но они почти не двигаются! Сделайте их быстрыми! Дайте мне ленивцев, которые похожи на бегунов!"

Теперь мы знаем, как удовлетворить это нелепое требование: найдите стрелку, идущую из левого нижнего угла к бегуну, как показано слева на roi 20 *4,* и прибавьте ее к голове стрелки, указывающей на ленивца, - получится бурый медведь. То есть лошадь - лиса + бегун = бурый медведь, как показано справа на *рис. 20-4*.

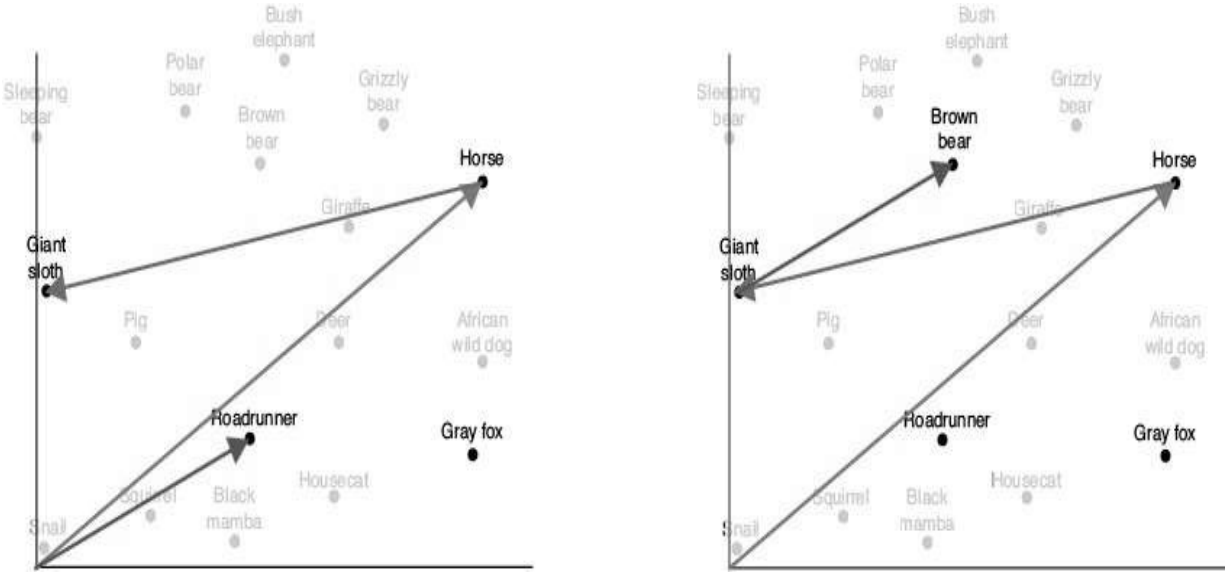

Рисунок 20-4: *Слева:* ***Мы можем нарисовать стрелку к бегуну. Справа:*** Гигантский ***ленивец*** + ***бегун-дорожник*** *- бурый медведь.*

Вы предлагаете директору группу бурых медведей (так называемый "*сонм* медведей"). Директор драматически закатывает глаза. "Наконец-то. Что-то быстрое, как лошади, но не дерганое, как лисы, и быстрое, как бегуны. Это только то, о чем я просил в первую очередь". Они снимают сцену погони с медведями, и фильм впоследствии

выходит с большим успехом.

В этой истории есть два ключевых элемента. Первый заключается в том, что животные на нашем графике были расположены полезным образом, хотя мы не знали, что это за способ, и что представляют собой оси в отношении данных.

Второй ключевой момент заключается в том, что метки осей нам в конечном итоге не понадобились. Мы могли перемещаться по графику, просто добавляя и вычитая стрелки, указывающие на элементы на самом графике. То есть мы не пытались найти "более медленную лошадь". Скорее, мы работали строго с самими животными, а их различные атрибуты привносились неявно. Убрав из такого крупного животного, как лошадь, быстроту лисы, мы получили крупное медленное животное.

Какое отношение это имеет к обработке языка?

Встраивание слов

Чтобы применить увиденное к словам, мы заменим животных словами. И вместо того, чтобы использовать только две оси, мы поместим наши слова в пространство с сотнями измерений.

Для этого используется алгоритм, который определяет, что должна означать каждая ось в этом пространстве, и помещает каждое слово в соответствующую точку. Вместо
Присваивая каждому слову одно число, алгоритм присваивает слову целый список чисел, представляющих его координаты в огромном пространстве.

Этот алгоритм называется ***эмбеддером,*** и мы говорим, что этот процесс представляет собой *встраивание* слов в *пространство встраивания,* тем самым создавая *эмбеддинги слов.*

Встраиватель сам решает, как построить пространство и найти координаты каждого слова так, чтобы оно находилось рядом с похожими словами. Например, если он видит много предложений, начинающихся со слов I just drank some, то любое существительное, следующее за ним, интерпретируется как некий вид напитка, и оно помещается рядом с другими видами напитков. Если он видит, что я только что съел красное, то все, что идет следом, интерпретируется как нечто красное и съедобное, и оно помещается рядом с другими вещами, которые являются красными и рядом с другими съедобными вещами. То же самое можно сказать о десятках и даже сотнях других связей, как очевидных, так и тонких. Поскольку пространство имеет очень много измерений, а оси могут иметь произвольно сложные значения, слова могут одновременно относиться ко многим кластерам, основанным на,

казалось бы, не связанных между собой характеристиках.

Эта идея является одновременно абстрактной и мощной, поэтому давайте проиллюстрируем ее на конкретных примерах

примеры. Мы попробовали несколько выражений "словесной арифметики", используя предварительно обученное встраивание из 684 754 слов, сохраненных в пространстве 300 измерений (авторы spaCy 2020). Первым тестом было известное выражение: king - man + woman (El Boukkouri 2018). Система выдала королеву как наиболее вероятный результат, что вполне логично: можно представить, что встраиватель отработал некий смысл знатности по одной оси и пола по другой. Другие тесты были близки, но не идеальны. Например, слово lemon - yellow + green (лимон - желтый + зеленый) дало наилучшее совпадение с имбирем, но ожидаемый лайм оказался не так уж далеко - пятым по близости словом. Аналогично, труба - клапаны + задвижка дали наиболее вероятный результат - саксофон, но ожидаемый тромбон занял первое место.

Прелесть обучения эмбеддера в пространстве с сотнями (или даже тысячами) измерений заключается в том, что он может использовать это пространство гораздо эффективнее, чем это мог бы сделать любой человек, что позволяет ему одновременно представлять огромное количество отношений.

Арифметика слов, которую мы только что рассмотрели, является забавной демонстрацией встраивания пространств, но она также позволяет нам осмысленно выполнять операции над словами, такие как сравнение, масштабирование и сложение, что важно для алгоритмов, рассматриваемых в этой главе.

Как только мы получили вкрапления слов, их легко включить практически в любую сеть. Вместо того чтобы присваивать каждому слову одно целое число, мы присваиваем ему вкрапление слова, которое представляет собой список чисел. Таким образом, вместо обработки тензоров нулевой размерности (одиночных чисел) система обрабатывает тензоры одномерной размерности (списки чисел).

Это позволяет решить проблему, которую мы наблюдали в главе 19, когда предсказания, близкие к цели, но не совсем верные, давали нам бессмыслицу. Теперь мы можем допустить некоторую неточность, поскольку похожие слова встраиваются рядом друг с другом. Например, мы можем задать нашей языковой модели фразу The dragon approached and let out a mighty, ожидая, что следующим словом будет roar. Алгоритм может предсказать тензор, который находится рядом с roar, но не совсем на нем, выдавая вместо него bellow или blast. Скорее всего, мы не получим в ответ что-то несвязанное, например, daffodil.

На рис. 20-5 показаны шесть наборов из четырех родственных слов, которые мы задали стандартному встраивателю слов. Чем больше похожи друг на друга вкрапления любых двух слов, тем выше оценка этой пары слов, и тем темнее их пересечение. График симметричен вокруг диагонали от левого верхнего до правого нижнего угла, поскольку порядок

сравнения слов не имеет значения.

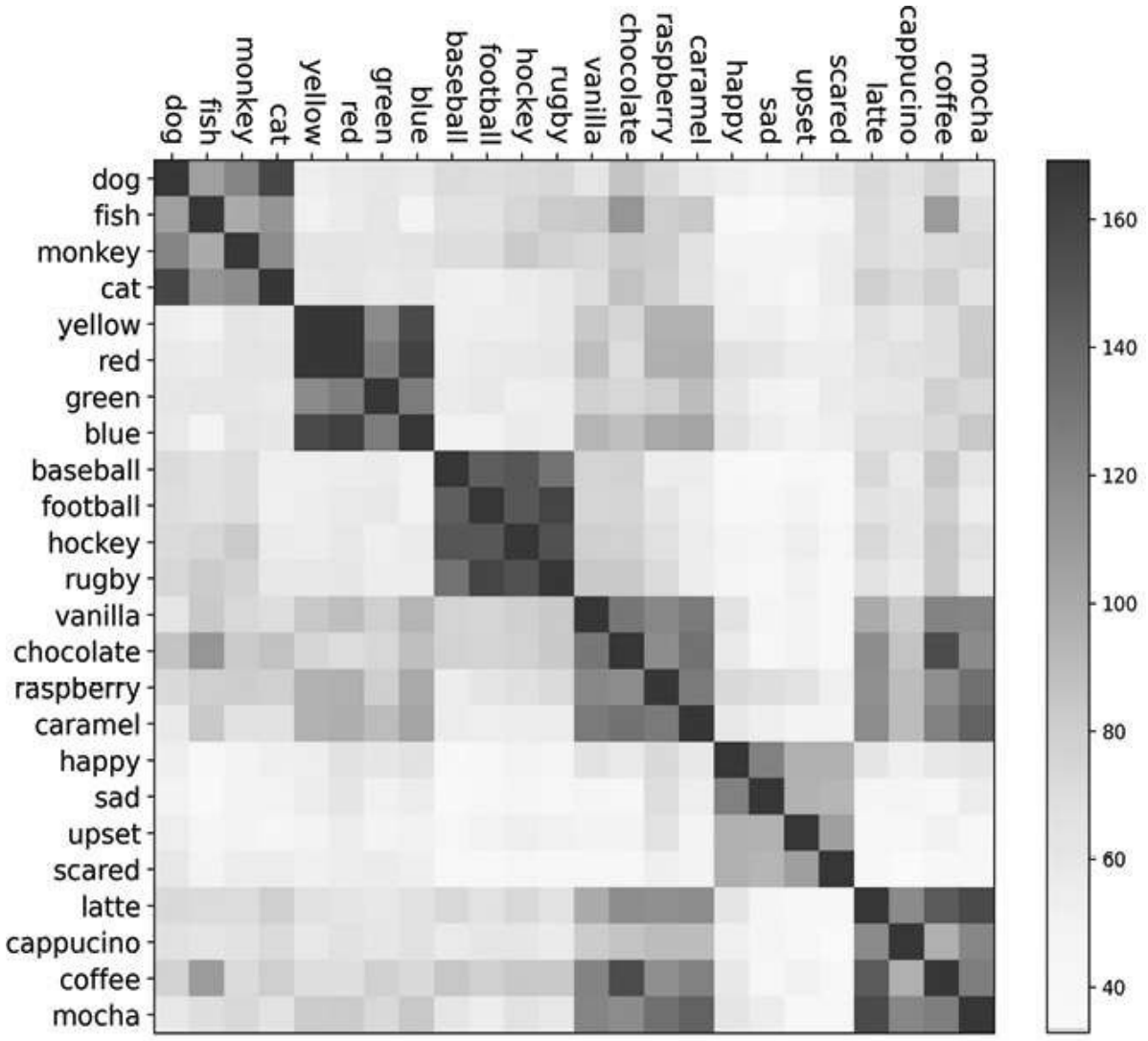

Рисунок 20-Ifi: Сравнение пар ***слов путем сравнения*** *сходства их вкраплений*

Из *рис. 20-5* видно, что каждое слово наиболее сильно совпадает с самим собой, а также более сильно совпадает с родственными словами, чем с неродственными. Поскольку мы разместили родственные слова рядом, на графике их сходство показано в виде небольших блоков. Однако есть несколько любопытных моментов. Например, почему рыба лучше, чем в среднем, сочетается с шоколадом и ко9и, а синий - с карамелью? Возможно, это анифакты панических обучающих данных, используемых для данного встраивателя.

Кофейные напитки и ароматизаторы хорошо сочетаются друг с другом, возможно, потому, что люди заказывают кофейные напитки с этими ароматизированными сиропами. Есть также намек на связь между цветами и вкусами.

Многие предварительно обученные вставки слов широко доступны бесплатно и легко загружаются практически в любую библиотеку. Мы можем просто импортировать их и сразу получить вектор для любого слова. Встраивания GLoVe (Mikolov et al. 2013a; Mikolov et al. 2013b) и word2vec (Pennington, Socher, and Manning 2014) использовались во многих проектах. Более поздний проект fastText (Facebook Open Source 2020) предлагает вкрапления на 157 языках.

Мы также можем встраивать целые предложения, чтобы сравнивать их как единое целое, а не слово за словом (Cer et al. 2018). На *рис. 20-6* показано сравнение вкраплений для десятка предложений (TensorFlow 2018). В этой книге мы сосредоточимся на вкраплениях слов, а не предложений.

Рисунок 20-6: Сравнение вкраплений предложений. Чем больше балл, тем больше предложения считаются похожими друг на друга.

ELMo

Встраивание слов - это огромный прогресс по сравнению с присвоением словам целых чисел. Но даже несмотря на мощный потенциал вкраплений слов, у подхода, который мы описали ранее для их создания, есть проблема: нюансы.

Как мы видели в главе 19, во многих языках есть слова с разными значениями, но пишутся и произносятся они одинаково. Если мы хотим понять смысл слов, нам необходимо различать эти значения. Один из способов сделать это - дать каждому значению слова свое вложение. Так, слово cupcake, имеющее одно значение, имеет одно вложение. А у слова train есть два вложения: одно для существительного (например, "я ехал на поезде") и одно для глагола (например, "я люблю дрессировать собак"). Эти два значения слова train действительно представляют собой совершенно разные идеи, в которых просто используется одна и та же последовательность букв.

Такие слова представляют собой две проблемы. Во-первых, необходимо создать уникальные вкрапления для каждого значения. Во-вторых, необходимо выбрать правильное вложение, когда такие слова используются в качестве входных данных. Решение этих задач требует учета контекста каждого слова. Первый алгоритм, который сделал это в значительной степени, называется Embedding from *Language Models,* но он более известен под дружественной аббревиатурой *ELMo* (Peters et al. 2018), которая является именем маппета из детской телепередачи "*Улица Сезам"*. Мы говорим, что ELMo производит *контекстуализированные вкрапления слов.*

Архитектура ELMo похожа на архитектуру пары би-РНС, которую мы рассматривали на *рис. 19-20,* но ее части организованы по-другому. В стандартной би-РНС мы объединяем две РНС, работающие в противоположных направлениях.

ELMo меняет эту ситуацию. Хотя в нем используются две сети RNN, работающие в прямом направлении, и две сети, работающие в обратном направлении, они сгруппированы по направлениям. Каждая из этих групп представляет собой RNN глубиной в два слоя, подобно той, что мы видели на *рис. 19-21*. Архитектура ELMo показана на *рис. 20-7*. Традиционно диаграммы ELMo рисуются в красной цветовой гамме, поскольку Элмо на улице *Сезам* - ярко-красный персонаж.

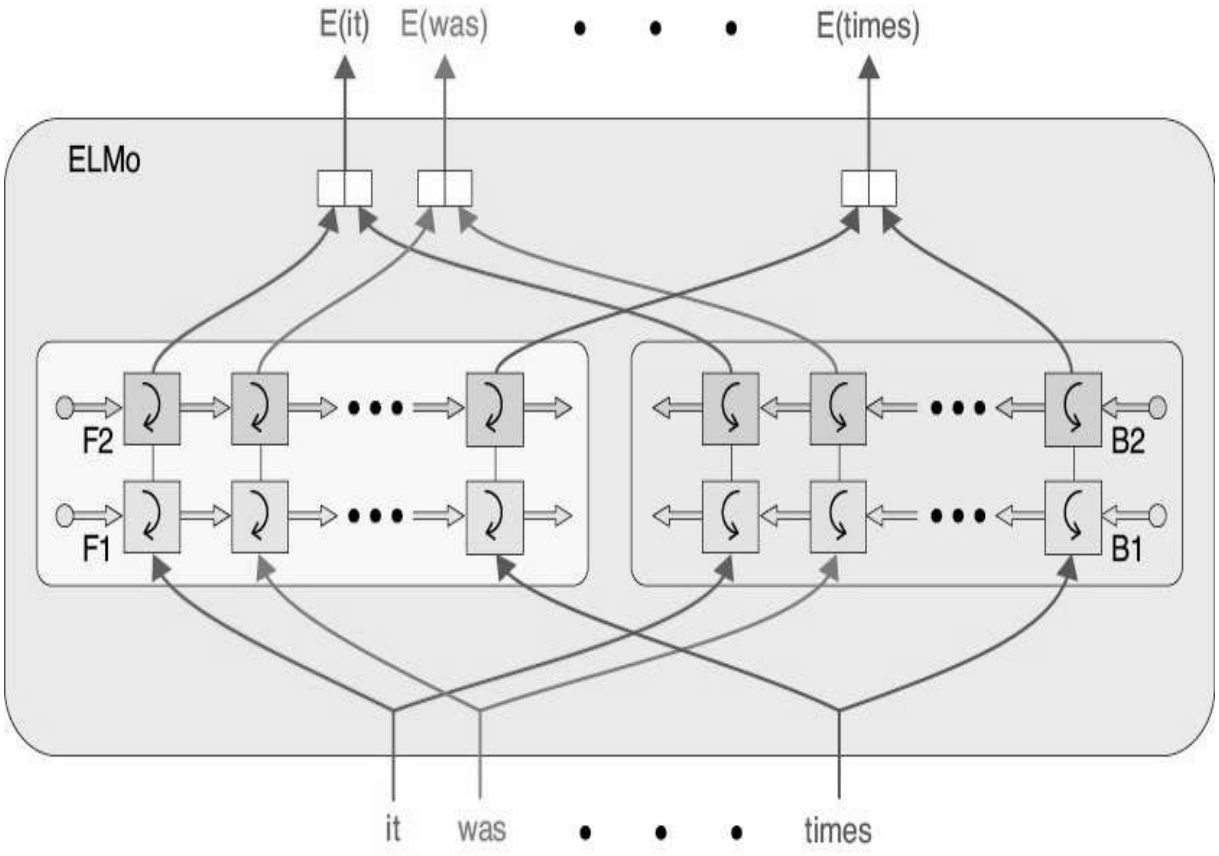

Рисунок 20-7: Структура ELMo в развернутом виде. Вводный текст находится* в *самом низу. Вложение каждого входного элемента находится вверху.

При такой архитектуре каждое входное слово превращается в два новых тензора: один из прямых сетей (обозначенных F1 и F2), учитывающий предшествующие слова, и один из обратных сетей (обозначенных B1 и B2), учитывающий последующие слова. Объединяя эти результаты, мы получаем контекстуальные вкрапления слов, учитывающие все остальные слова в предложении.

Обученные версии ELMo широко доступны для бесплатного скачивания в различных объемах (Gluon 2020). Как только мы получили предварительно обученный ELMo, его легко использовать в любой языковой модели. Мы передаем ELMo все предложение, а в ответ получаем контекстуализированное вложение слова для каждого слова с учетом его контекста.

На рис. 20-8 показаны четыре предложения, в которых омоним train используется в качестве глагола, и четыре предложения, в которых train используется в качестве существительного. Мы предоставили их стандартной модели ELMo, обученной на базе данных из 1 млрд. слов, которая помещает каждое слово в пространство из 1024 измерений (TensorFlow 2020a). Мы извлекли вложение ELMo слова train в каждое предложение и сравнили его вложение со вложением слова train в

во всех остальных предложениях. Несмотря на то, что слово написано одинаково в каждом предложении, ELMo может определить правильное вложение, основываясь на контексте слова.

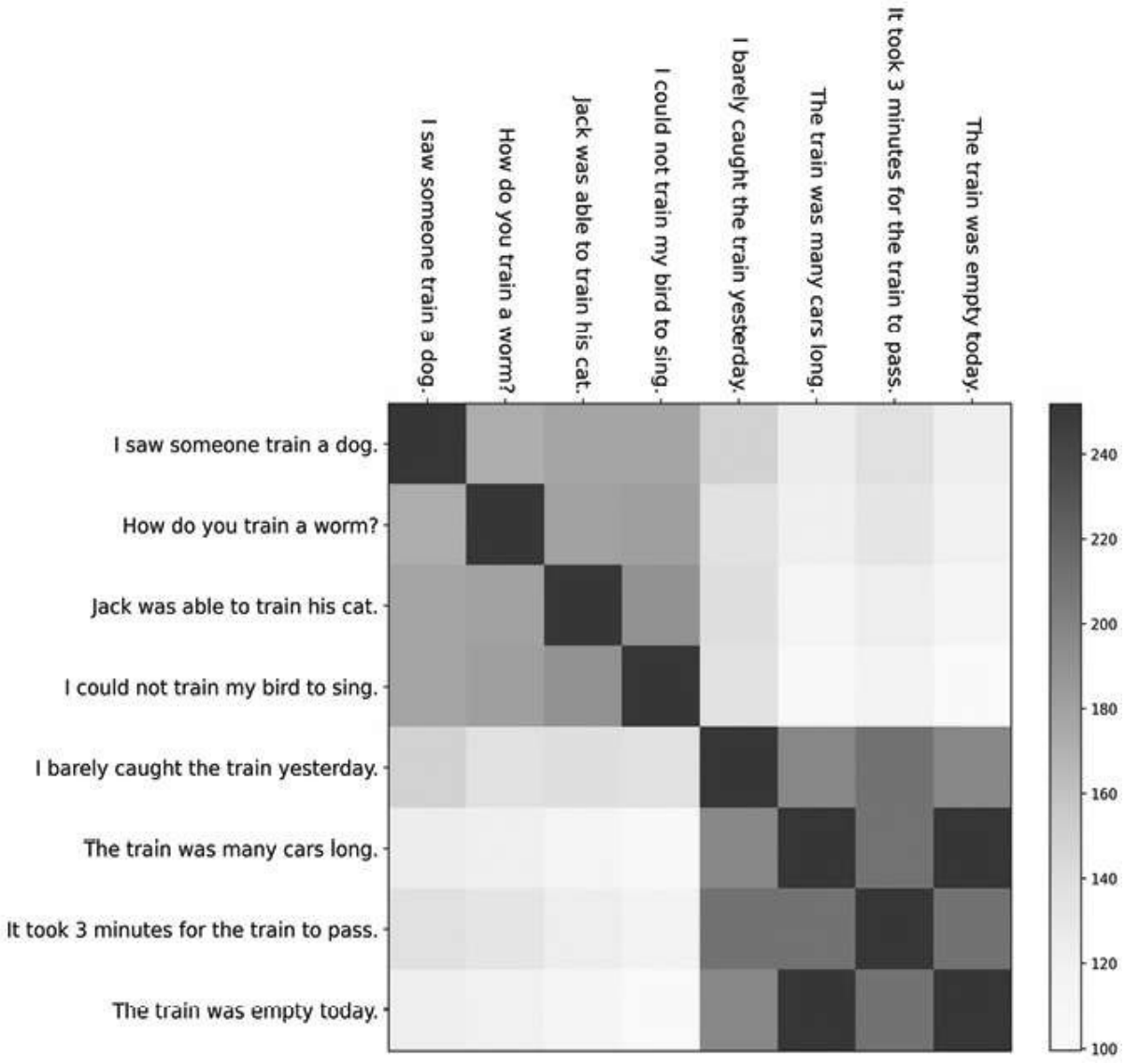

Рис. 20-8: Сравнение вложений ELMo в поезд*, полученных в р е з у л ь т а т е его использования в различных предложениях. Более темные цвета означают большее сходство вкраплений.*

Обычно мы размещаем алгоритмы встраивания, такие как ELMo, на собственном слое в системе глубокого обучения. Часто это самый первый слой в сети обработки языка. Наш значок для алгоритма встраивания, показанный на сайте Froi 20*9*, предлагает взять пространство слов и поместить его в более широкое пространство встраивания.

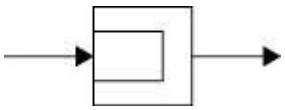

Рисунок 20-9: Наша иконка для встраивания layer

ELMo и другие подобные ему алгоритмы, такие как ***Universal Language Model Fine-Tuning,*** или ***ULMFiT*** (Howard and Ruder 2018), обычно обучаются на базах данных общего назначения, таких как книги и документы из Интернета. Когда они нужны для решения какой-то конкретной задачи, например, медицинской или юридической, мы обычно дорабатываем их, используя дополнительные примеры из этих доменов. В результате получается набор вкраплений, включающих специализированный язык этих областей, сгруппированных по их особым значениям в жаргоне.

Мы будем использовать эмбеддинги в системах, которые построим позже в этой главе. Эти сети будут опираться на механизм внимания, так что давайте рассмотрим его сейчас.

Внимание

В главе 19 мы рассмотрели, как улучшить перевод, принимая во внимание все слова в предложении. Но когда мы переводим конкретное слово, не все слова в предложении одинаково важны или даже уместны.

Например, предположим, что мы переводим предложение I saw большая собака ела свой обед. При переводе слова dog нам, скорее всего, не важно слово saw, но для правильного перевода местоимения his может потребоваться связать его с двумя словами big dog.

Если для каждого слова в исходном тексте определить, какие еще слова могут повлиять на перевод, то можно сосредоточиться только на этих словах и игнорировать остальные. Это даст значительную экономию как памяти, так и времени вычислений. И если мы сможем решить эту задачу так, чтобы не зависеть от последовательной обработки слов, то это можно будет делать даже параллельно.

Алгоритм, который выполняет эту работу, называется *вниманием,* или *set[-attention* (Bahdanau, Cho, and Bengio 2016; Sutskever, Vinyals, and Le 2014; Cho et al. 2014). Внимание позволяет сфокусировать наши ресурсы только на тех частях входного сигнала, которые имеют значение.

Современные версии внимания часто основаны на технике, называемой *запросом, ключом, значением* или просто QfV. Эти термины пришли из области баз данных и могут показаться несколько непонятными в данном контексте. Поэтому мы будем описывать эти понятия с

помощью

различные наборы терминов и затем соединить их в конце с запросом, ключом и значением.

Кфоризационная *аналогия*

Начнем с аналогии. Предположим, что вам нужно купить краску, но вам сказали, что цвет должен быть "светло-желтым с небольшими вкраплениями темно-оранжевого".

В единственном в городе магазине красок единственный продавец работает недавно и не знаком с красками. Вы оба предполагаете, что для получения нужного цвета вам придется смешать несколько стандартных красок, но не знаете, какие краски выбрать и сколько каждой из них использовать.

Продавец предлагает сравнить описание желаемого цвета с названиями цветов на каждой банке с краской. Возможно, некоторые названия совпадут лучше, чем другие. Продавец ставит воронку на пустую банку и предлагает налить в нее немного краски из каждой банки, стоящей на полках, ориентируясь на то, насколько название этой банки соответствует вашему описанию. То есть вы будете сравнивать желаемое описание "светло-желтый с небольшими вкраплениями темно-оранжевого" с тем, что напечатано на этикетке каждой банки, и чем лучше совпадение, тем больше краски вы нальете в воронку.

На рис. 20-10 наглядно показана идея для шести банок с краской. Здесь показаны их названия и качество совпадения каждого названия с описанием желаемого цвета. Мы получили хорошие совпадения по "Sunny Yellow" и "Orange Crush", хотя немного "Lunch with Teal" проскользнуло благодаря совпадению со словом "with".

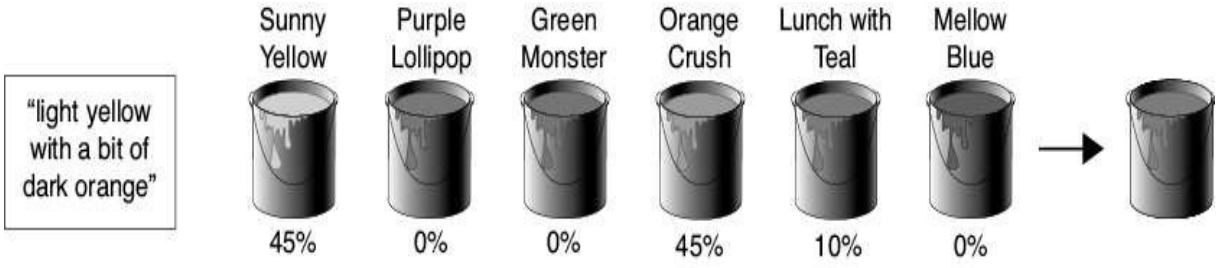

Рис. 20-10: Получив описание цвета (слева), мы комбинируем некоторые из них, основываясь на том, *насколько их название соответствует описанию (в середине), и получаем окончательный результат (справа).*

В этой истории есть три момента, на которых следует сосредоточиться.

Во-первых, это ваша *просьба:* "светло-желтый с небольшим количеством темно-оранжевого". Во-вторых, есть *описание* на каждом

например, "Sunny Yellow" или "Mellow Blue". В-третьих, это содержание краски, которая действительно находится в каждой банке. По сюжету вы сравнивали свой запрос с описанием каждой банки, чтобы определить, насколько они совпадают. Чем лучше совпадение, тем больше содержимого банки вы использовали в конечной смеси.

Вкратце это и есть внимание. Задав запрос, сравните его с описанием каждого возможного элемента и включите часть содержания каждого элемента в зависимости от того, насколько его описание соответствует запросу.

Авторы первой работы о внимании сравнили этот процесс с обычным типом транзакций, используемых в базах данных. На языке баз данных мы ищем что-то, посылая *запрос* в базу данных. В таком процессе каждый объект в базе данных имеет описательный *ключ,* который может отличаться от фактического *значения* объекта. Заметим, что под словом "*значение"* здесь понимается содержимое объекта, будь то одно число или что-то более сложное, например строка или тензор.

Система баз данных сравнивает запрос (или запрос) с каждым ключом (или описанием) и использует эту оценку для определения того, какую часть ценности (или содержания) объекта следует включить в конечный результат. Таким образом, наши понятия "запрос", "описание" и "содержание" соответствуют понятиям "запрос", "ключ" и "значение", или, чаще, QKV.

Самоучитель

На рис. 20-11 показана фундаментальная операция внимания в абстрактной форме. Здесь мы имеем пять слов. Каждая из трех цветных коробок представляет собой небольшую нейронную сеть, которая принимает числовое представление слова и преобразует его в новое (часто эти сети представляют собой всего один полностью связанный слой). В данном примере мы хотим перевести слово "собака". Поэтому нейронная сеть (красная) преобразует тензор для слова dog и превращает его в новый тензор, представляющий запрос Q. Как показано на рисунке, еще две небольшие нейронные сети преобразуют тензор для слова dinner в новые тензоры, соответствующие его ключу K (из синей сети) и его значению V (из зеленой сети).

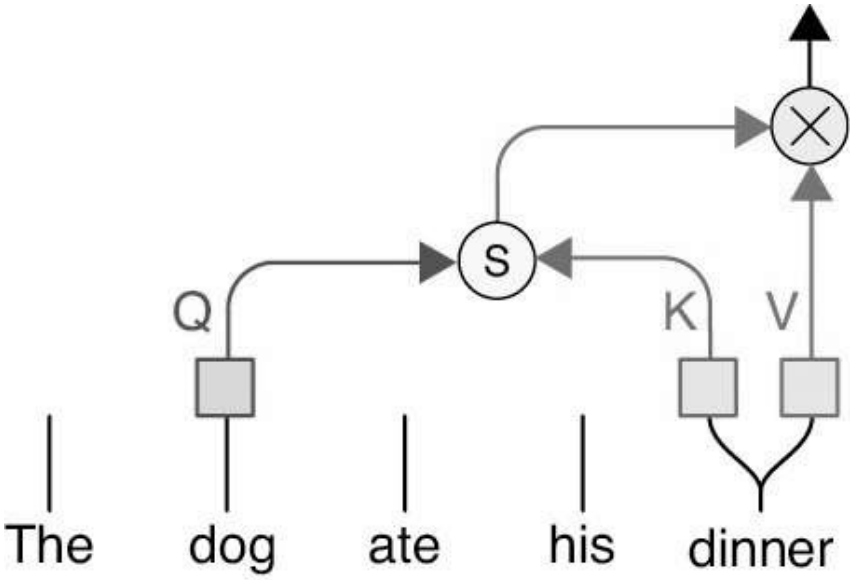

Рисунок 20-11: Основной шаг внимания, использующий в качестве запроса слово dog *для определения релевантности слова* dinner. *Каждый блок представляет собой небольшую нейронную сеть, которая преобразует входные данные* в *запрос, ключ или значение.*

На практике мы сравниваем запрос dog с ключом каждого слова в предложении, включая саму собаку. В данной иллюстрации мы ограничимся сравнением со словом dinner.

Мы сравниваем запрос и ключ, чтобы определить, насколько они похожи. Для этого мы используем небольшую функцию подсчета баллов, которую обозначаем буквой *S* в круге.
Не вдаваясь в математику, эта функция сравнивает два тензора и выдает одно число. Чем больше тензоры похожи друг на друга, тем больше это число. Функция скоринга обычно предназначена для получения числа в диапазоне
0 и 1, причем большие значения свидетельствуют о лучшем соответствии.

Мы используем результат скоринговой функции для масштабирования тензора, представляющего значение ужина. Чем больше совпадений между запросом и ключом, тем больше результат шага масштабирования, и тем большее значение ужина попадет в результат.

Посмотрим, как это выглядит, если применить этот фундаментальный шаг одновременно ко всем словам на входе. Продолжим рассмотрение перевода слова dog. Общий результат представляет собой сумму отдельных масштабированных значений всех входных слов. На *рисунке* ***20-12*** показано, как это выглядит.

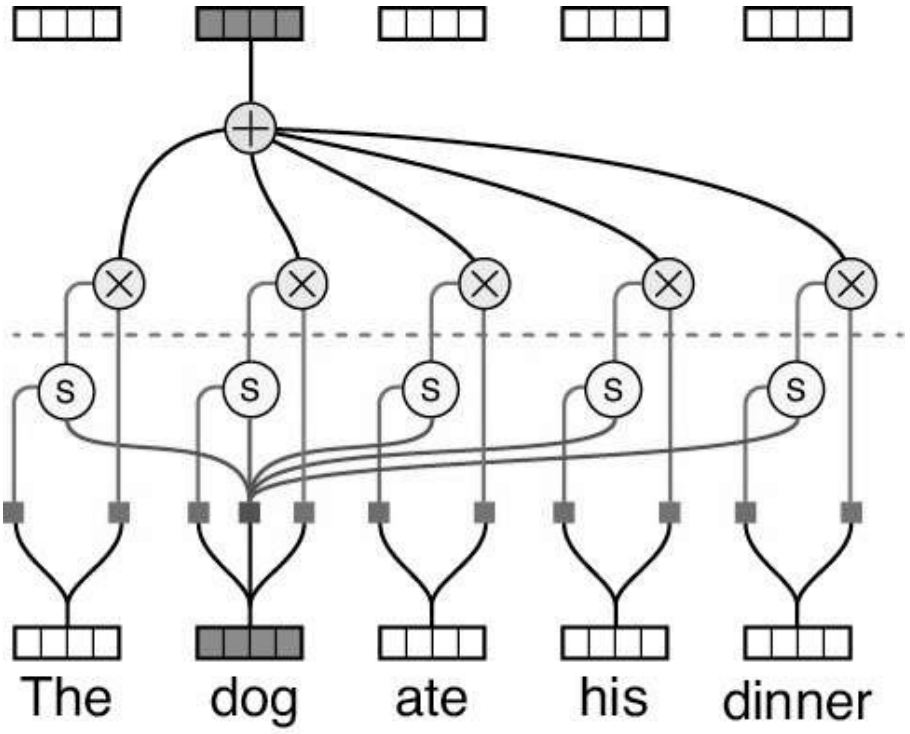

Рис. 20-12: Использование внимания для одновременного определения вклада all *пяти слов в предложении в слово* собака. *Пространственное и цветовое кодирование ОКВ соответствует рис. 20-11. На рисунке все* дафы *текут вверх.*

На рис. 20-12 следует отметить несколько моментов. Во-первых, здесь задействованы только три нейронные сети - по одной для вычисления тензоров запросов, ключей и значений. Мы используем одну и ту же сеть "вход - запрос" (красная на рисунке) для преобразования каждого входного сигнала в запрос, одну и ту же сеть "вход - ключ" (синяя на рисунке) для преобразования каждого входного сигнала в ключ и одну и ту же сеть "вход - значение" (зеленая на рисунке) для преобразования каждого входного сигнала в его значение. Эти преобразования необходимо применять только один раз для каждого слова.

Во-вторых, после оценок и перед масштабированием значений имеется пунктирная линия. Она представляет собой шаг softmax, применяемый к оценкам, за которым следует деление. Эти две операции не позволяют числам, полученным в результате подсчета, стать слишком большими или маленькими. Кроме того, sohmax преувеличивает влияние близких матчей.

В-третьих, мы суммируем все масштабированные значения, чтобы получить новый тензор для dog, в том числе и от значения самого dog. Часто оказывается, что каждое слово имеет наибольшее количество баллов по отношению к самому себе. Это не так уж плохо, поскольку в данном случае наиболее важным словом для перевода dog действительно является само dog. Но бывают случаи, когда другие слова

будет иметь большее значение. В качестве примера можно привести случаи, когда меняется порядок слов, когда слово не имеет прямого перевода и должно опираться на другие слова, или когда мы пытаемся разрешить местоимение.

Четвертый важный момент заключается в том, что мы применяем обработку *рис. 20-12* одновременно ко всем словам во входном предложении. То есть каждое слово рассматривается как запрос, и весь процесс выполняется независимо для этого слова, как показано на *рис. 20-13*.

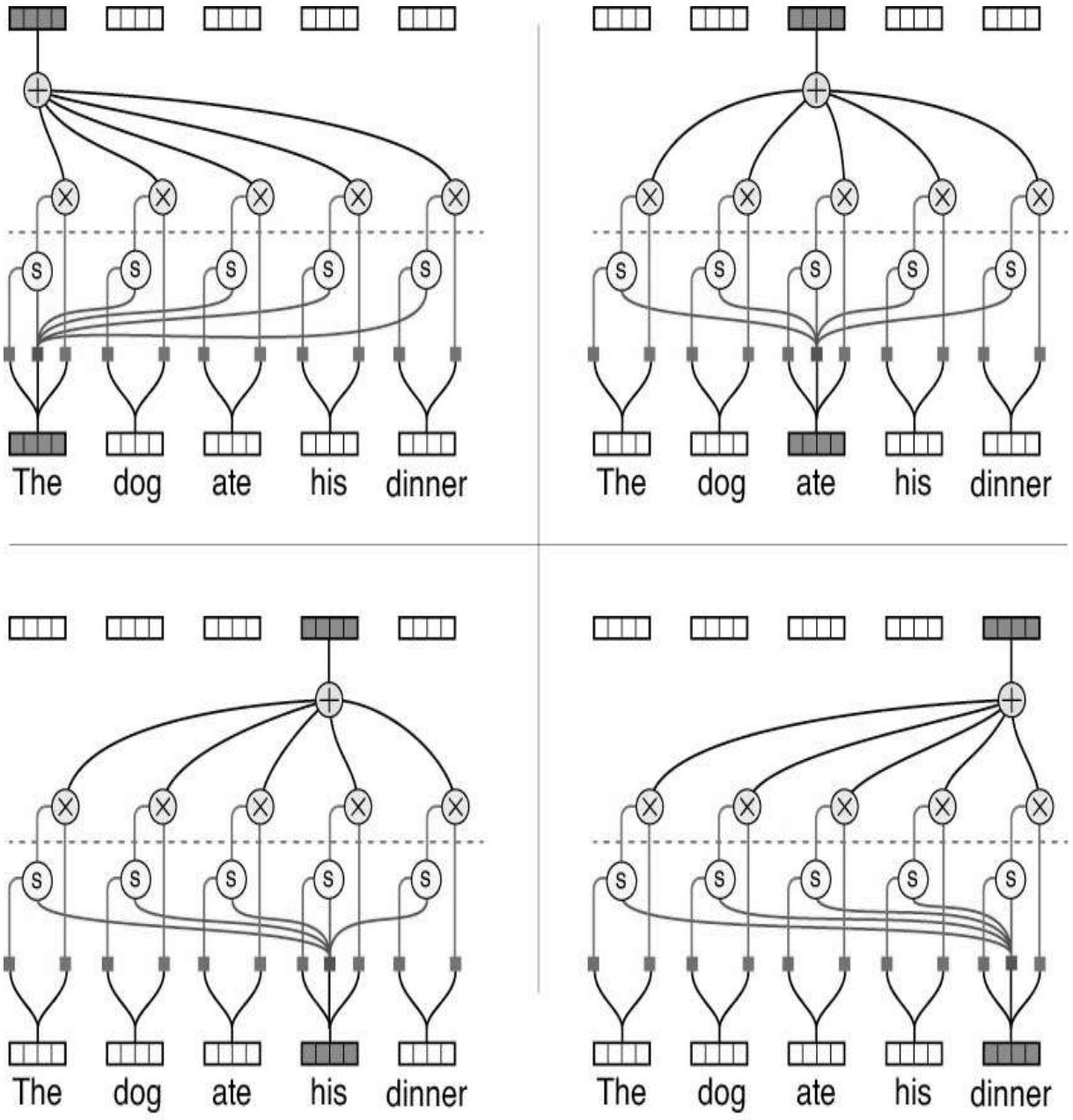

Рисунок 20-13: Применение внимания к остальным четырем словам в нашем предложении

Наш пятый и последний пункт - это просто явное повторение того, на что мы все время обращали внимание: вся обработка, представленная на *рис. 20-12* и *рис. 20-13*, может быть выполнена параллельно всего за четыре шага, независимо от длины предложения. На шаге 1 входные данные преобразуются в тензоры запросов, ключей и значений. На шаге 2 все запросы и ключи оцениваются в баллах. Шаг 3 использует оценки для масштабирования значений, а шаг 4 складывает масштабированные значения для получения нового выходного результата для каждого входа.

Ни один из этих этапов не зависит от длины входного сигнала, поэтому мы можем обрабатывать длинные предложения за то же время, что и короткие, при условии, что у нас есть необходимая память и вычислительная мощность.

Мы называем процесс на *рис. 20-12* и *рис. 20-13 sel[-attentiOn,* поскольку механизм внимания использует один и тот же набор входов для вычисления всего: запросов, ключей и значений. То есть мы выясняем, насколько входные данные должны быть внимательны к себе.

Когда мы помещаем самовнимание в глубокую сеть, мы помещаем его на собственный *слой самовнимания, который* часто называют просто *слоем внимания.* На вход подается список слов в числовом виде, на выходе получается то же самое.

Двигателями внимания являются скоринговая функция и нейронные сети, преобразующие входные данные в запросы, ключи и значения. Рассмотрим их вкратце.

Функция скоринга сравнивает запрос с ключом, возвращая значение от 0 до 1, причем чем более похожи два значения, тем выше их оценка. Значит, каким-то образом входы, которые мы считаем похожими, должны иметь похожие значения в функции подсчета баллов. Теперь мы можем увидеть практическую ценность вкраплений.
Вспомните, как мы обсуждали "*Повесть о* двух *городах"* в главе 19, где мы присваивали каждому слову номер, определяемый его порядковым номером в тексте. В результате слова keep и flint получили номера 1,003 и 1,004 соответственно. Если бы мы просто сравнили эти числа, то они получили бы высокую оценку сходства. Для большинства предложений это не то, что нам нужно. Если мы используем значение запроса для глагола keep, то обычно хотим, чтобы оно было похоже на ключи для таких синонимов, как retain, hold, reserve, и совсем не похоже на ключи для таких несвязанных слов, как flint, preposterous или dinosaur. Вкрапления - это средства, с помощью которых похожие слова (или слова, используемые в похожих значениях) получают похожие представления.

Необходимая тонкая настройка вкраплений - задача нейронной системы.

сети, которые преобразуют входные слова в представления, где они могут быть осмысленно сопоставлены в контексте предложения, в котором они используются. Единственная причина, по которой у нас есть хоть какой-то шанс на это, заключается в том, что слова уже встроены в пространство, где похожие слова находятся рядом друг с другом.

Аналогичным образом, задача сети, превращающей входные данные в значения, состоит в том, чтобы представить эти значения таким образом, чтобы их можно было удобно масштабировать и комбинировать. Смешивание двух встроенных слов дает слово, находящееся между ними.

QfKV AHention

В сети самовнушения, представленной на *рис. 20-12,* запросы, ключи и значения формируются из одних и тех же входных данных, что и привело к названию "самовнушение".

Популярный вариант использует один источник для запросов, а другой - для ключей и значений. Это более соответствует нашей аналогии с магазином красок, где мы приходим с запросом, а магазин располагает ключами и значениями. Мы называем этот вариант сетью Q/KV, где косая черта указывает на то, что запросы поступают из одного источника, а ключи и значения - из другого. Этот вариант иногда используется при добавлении внимания к сети типа seq2seq, где запросы поступают от кодера, а ключи и значения - от декодера, поэтому его также иногда называют слоем *внимания кодера-декодера*. Структура показана на *рис. 20-14*.

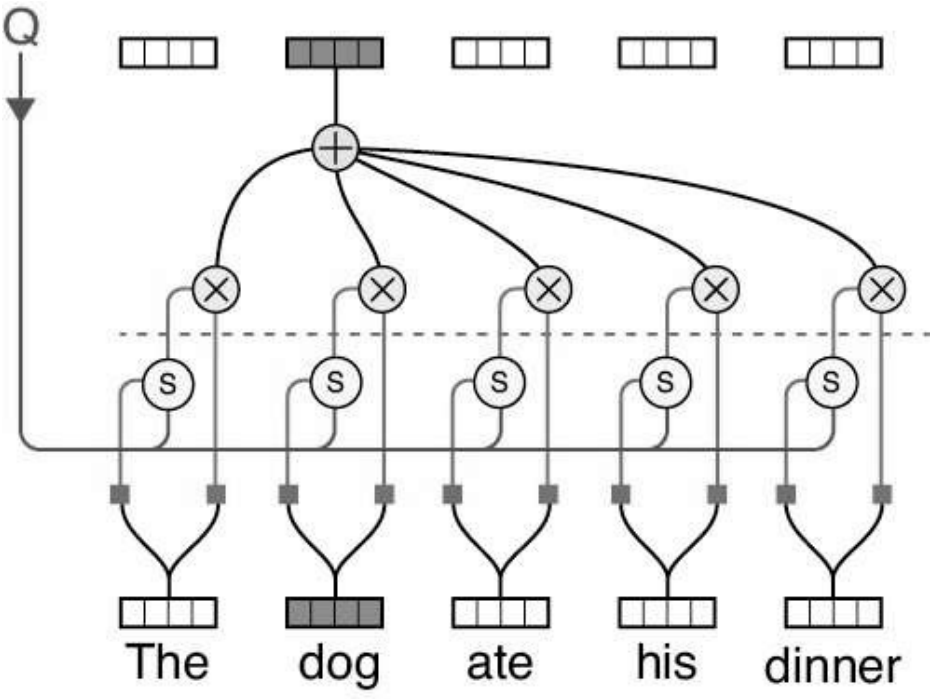

Рисунок 20-14: Слой Q/KV похож на самовнимание, как показано на рис. 20-12, за исключением того, что запросы поступают не со входов.

Многоголовое внимание

Идея внимания состоит в том, чтобы выявить слова, которые похожи друг на друга, и создать из них полезную смесь. Однако слова можно считать похожими по разным параметрам. Мы можем считать похожими существительные, или цвета, или пространственные понятия, такие как "вверх" и "вниз", или временные понятия, такие как "вчера" и "завтра". Что из этого лучше выбрать?

Разумеется, единого оптимального ответа не существует. На самом деле мы часто хотим сравнивать слова сразу по нескольким критериям. Например, при написании текста песни мы можем поставить высокие баллы парам слов, которые имеют схожие значения, схожие звуки в последнем слоге, одинаковое количество слогов и одинаковое ударение в слогах. Когда мы пишем о спорте, мы можем вместо этого сказать, что игроки в одних командах и с одинаковыми амплуа похожи друг на друга.

Мы можем оценивать слова по нескольким критериям, просто запуская одновременно несколько независимых сетей внимания. Каждая сеть называется "*головой*". Инициализируя каждую голову независимо, мы надеемся, что в процессе обучения каждая голова научится сравнивать входные данные по критериям, которые одновременно полезны и отличаются от критериев, используемых другими слоями. При желании можно добавить дополнительную обработку, чтобы явно побудить разные головы обратить внимание на разные аспекты входных данных. Эта идея называется *многоголовочным вниманием,* и мы можем применить ее как к сетям самовнимания, как на *рис. 20-12*, так и к сетям Q/KV, как на *рис. 20-14*.

Каждая голова представляет собой отдельную сеть внимания. Чем больше голов, тем на большем количестве различных аспектов входного сигнала они могут сосредоточиться.

Схема слоя внимания с несколькими головками показана на *рис. 20-15*. Как видно из рисунка, обычно мы объединяем выходы головок в список и пропускаем его через один полностью связанный слой. Таким образом, выход всей многоголовочной сети имеет ту же форму, что и вход. Такой подход позволяет легко размещать несколько многоголовочных сетей друг за другом.