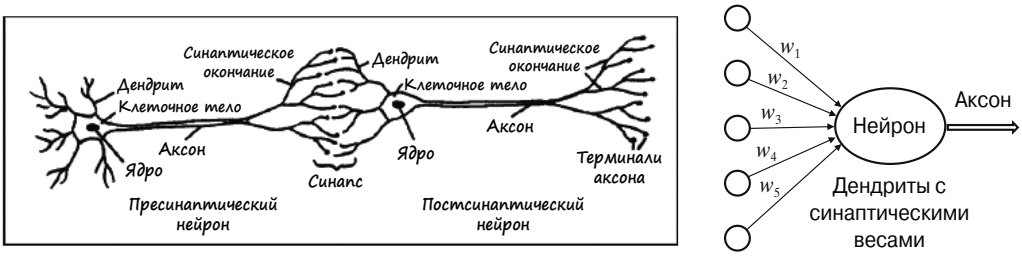
Глава 1

Введение в нейронные сети

Не сотвори машину, выдающую себя за человека.
Фрэнк Герберт

1.1. Введение

Термин *искусственные нейронные сети* охватывает совокупность популярных методов машинного обучения, имитирующих механизмы обучения, которые действуют в биологических организмах. Нервная система человека содержит *нейроны* — специфические клетки, предназначенные для приема, обработки, хранения и передачи поступающей информации с помощью электрических и химических сигналов. Нейроны могут соединяться друг с другом посредством *аксонов* и *дендритов*, а структуры, связывающие дендриты одного нейрона с аксонами других нейронов, называются *синапсами* (рис. 1.1, *а*). Сила синаптических связей часто изменяется в ответ на внешнее возбуждение. Именно эти изменения лежат в основе механизма обучения живых организмов.



а) Биологическая нейронная сеть

б) Искусственная нейронная сеть

Рис. 1.1. Синаптические связи между нейронами; изображение слева взято из курса "The Brain: Understanding Neurobiology Through the Study of Addiction" [598]

Описанный биологический механизм имитируют *искусственные* нейронные сети, которые содержат вычислительные элементы, также получившие название *нейроны*. На протяжении всей книги мы будем использовать термин "нейронные сети", подразумевая искусственные нейронные сети, а не биологические. Вычислительные элементы соединяются между собой посредством связей, имеющих определенные веса, которые играют ту же роль, что и силы синаптических связей в биологических организмах. Каждый вход нейрона масштабируется в соответствии с его весом, что влияет на результат вычисления функции в каждом элементе (рис. 1.1, *б*). Искусственная нейронная сеть вычисляет нелинейную функцию активации от взвешенной суммы входов, распространяя вычисленные значения от входных нейронов к выходным, причем веса играют роль промежуточных параметров. Процесс обучения происходит путем изменения весов связей, соединяющих нейроны. Точно так же, как для обучения биологических систем нужны внешние стимулы, в искусственных нейронных сетях внешним стимулом являются тренировочные данные, содержащие пары значений "вход — выход" той функции, которой должна обучиться сеть. *Тренировочными (учебными) данными* могут быть пиксельные представления изображений (вход) и их аннотационные метки (например, *морковь*, *банан*), представляющие ожидаемый выход. Нейронная сеть использует тренировочные данные, передаваемые ей в качестве входных обучающих примеров, для предсказания (прогнозирования) выходных меток. Тренировочные данные обеспечивают обратную связь, позволяющую корректировать веса в нейронной сети в зависимости от того, насколько хорошо выход, предсказанный для конкретного входа (например, вероятность того, что на картинке изображена морковь), соответствует ожидаемой аннотационной выходной метке, определенной в составе тренировочных данных. Ошибки, допущенные нейронной сетью в процессе вычисления функции, можно уподобить неприятным ощущениям, испытываемым биологическим организмом, которые управляют изменением силы синаптических связей. Аналогично этому в нейронной сети веса связей между нейронами подстраиваются так, чтобы уменьшить ошибку предсказания. Целью изменения весов является изменение вычисляемой функции в направлении, обеспечивающем получение более точных предсказаний на будущих итерациях. Настройка весов, уменьшающая ошибку вычислений для конкретного входного примера, осуществляется в соответствии с определенной математически обоснованной процедурой. Благодаря согласованной настройке весов связей между нейронами на множестве пар "вход — выход" функция, вычисляемая нейронной сетью, постепенно улучшается настолько, что выдает все более точные предсказания. Поэтому, если для тренировки нейронной сети используется набор различных изображений бананов, то она в конечном счете

приобретает способность правильно распознавать бананы на изображениях, которые до этого ей не встречались. Способность точно вычислять функции незнакомых входов после прохождения тренировки (обучения) на конечном наборе пар "вход — выход" называется *обобщением модели* (model generalization). Именно способность нейронной сети к обобщению знаний, приобретенных с помощью предоставленных ей тренировочных данных, на незнакомые примеры, является основным фактором, определяющим полезность любой модели машинного обучения.

Попытки сравнения моделей искусственных нейронных сетей с биологическими моделями нередко подвергаются критике на том основании, что они являются не более чем жалким подобием действительных процессов, происходящих в человеческом мозге. Тем не менее принципы нейронаук часто используются в качестве ориентиров при проектировании архитектур нейронных сетей. Другая точка зрения состоит в том, что нейронные сети строятся как высокоуровневые абстракции классических моделей, широко применяемых в машинном обучении. Действительно, идея базовых вычислительных элементов в нейронных сетях была подсказана традиционными алгоритмами машинного обучения, такими как *регрессия по методу наименьших квадратов* и *логистическая регрессия*. Нейронные сети обретают свою мощь благодаря объединению многих вычислительных элементов и совместному обучению их весов для минимизации ошибки предсказания. С этой точки зрения нейронную сеть можно рассматривать как *вычислительный граф* элементов, повышение вычислительной мощности которых достигается за счет их объединения определенными способами. Если нейронная сеть используется в своей простейшей форме, без организации взаимодействия многочисленных вычислительных элементов между собой, то алгоритм обучения часто сводится к классическим моделям машинного обучения (об этом мы поговорим в главе 2). Реальное превосходство нейронной модели по сравнению с классическими подходами раскрывается в полную мощь лишь в случае объединения разрозненных вычислительных элементов в единое целое, когда веса моделей согласованно обучаются с учетом зависимостей между ними. Комбинируя множество вычислительных элементов, можно расширить возможности модели до уровня, обеспечивающего обучение более сложным функциям, чем те, которые свойственны элементарным моделям простого машинного обучения (рис. 1.2). Способы объединения этих элементов также оказывают влияние на вычислительные возможности архитектуры и требуют тщательного изучения и адекватной оценки аналитиков. К тому же для обучения возросшего количества весов в таких расширенных вычислительных графах требуются достаточно большие объемы тренировочных данных.

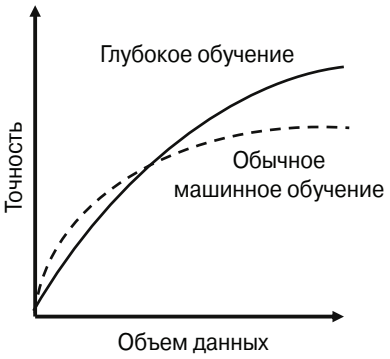


Рис. 1.2. Иллюстративное сравнение точности типичного алгоритма машинного обучения и точности большой нейронной сети. Преимущества методов глубокого обучения по сравнению с обычными методами проявляются главным образом при наличии достаточно больших объемов данных и вычислительных мощностей. В последние годы доступ к этим ресурсам значительно упростился, что привело к "Кембрийскому взрыву" в глубоком обучении

1.1.1. Человек и компьютер: расширение пределов возможностей искусственного интеллекта

Люди и компьютеры в силу самой своей природы приспособлены для решения задач разного типа. Например, для компьютера извлечение кубического корня из большого числа не составит труда, тогда как человеку справиться с этим нелегко. С другой стороны, такая задача, как распознавание объектов на изображении, для человека — сущий пустяк, тогда как для алгоритмов машинного обучения она долгое время оставалась трудным орешком. Лишь в последние годы при решении некоторых задач такого типа глубокое обучение продемонстрировало точность, недоступную для человека. Действительно, последние результаты, демонстрирующие превосходство алгоритмов глубокого обучения над человеком [184] при решении специализированных задач по распознаванию образов, еще каких-то десять лет назад считались бы невероятными большинством специалистов в области компьютерного зрения.

Многие архитектуры глубокого обучения, продемонстрировавшие столь необычайную эффективность, создавались вовсе не путем неструктурированного объединения вычислительных элементов. Исключительная эффективность *глубоких нейронных сетей* отражает тот факт, что истоки возможностей биологических нейронных сетей также кроются в их глубине. Более того, способы связывания биологических нейронных сетей между собой нам еще не до конца понятны. В тех немногих случаях, когда нам удалось в какой-то степени достигнуть понимания биологических структур, применение их в качестве

прототипов при проектировании искусственных нейронных сетей обеспечило значительный прорыв в этом направлении. В качестве классического примера архитектуры такого типа можно привести использование *сверточных нейронных сетей* для распознавания образов. Их архитектура была подсказана результатами экспериментов по исследованию организации нейронов в зрительной коре кошек, полученными Хубелем и Визелем в 1959 году [212]. Предшественником сверточной нейронной сети был *неокогнитрон* [127], архитектура которого основана непосредственно на этих результатах.

Структура нейронных связей человека совершенствовалась в процессе эволюции на протяжении миллионов лет, обеспечивая выживание вида. Выживание тесно связано с нашей способностью объединять чувства и интуицию в единое целое способом, который в настоящее время невозможно реализовать с помощью машин. Биологические нейронауки [232] все еще находятся на младенческой стадии развития, и мы располагаем лишь ограниченным набором сведений относительно того, как на самом деле работает человеческий мозг. Поэтому есть все основания полагать, что когда мы будем знать об этом больше, успех сверточных нейронных сетей, идея которых была подсказана биологическими моделями, будет повторен в других конфигурациях искусственных нейронных сетей [176]. Ключевым преимуществом нейронных сетей по сравнению с традиционным машинным обучением является то, что они обеспечивают возможность высокоуровневого абстрактного выражения семантики внутренних связей между данными посредством выбора вариантов архитектурных решений в вычислительном графе. Другое преимущество нейронных сетей заключается в том, что они позволяют очень легко регулировать сложность модели, добавляя или удаляя нейроны из архитектуры в соответствии с доступностью тренировочных данных или вычислительных ресурсов. Значительная доля недавних успехов нейронных сетей объясняется облегчением доступа к данным и ростом вычислительной мощности современных компьютеров, что позволило преодолеть ограничения традиционных алгоритмов машинного обучения, которые не в состоянии в полной мере воспользоваться всеми преимуществами доступных возможностей (см. рис. 1.2). В некоторых случаях традиционные алгоритмы машинного обучения работают лучше для небольших наборов данных в силу наличия большего количества вариантов выбора, более простых возможностей интерпретации моделей и распространенной тенденции вручную настраивать интерпретируемые признаки, вбирающие в себя специфические для данной предметной области внутренние закономерности. При ограниченных объемах данных лучшая из широкого разнообразия моделей машинного обучения обычно будет работать лучше, чем одиночный класс моделей (типа нейронных сетей). В этом и кроется одна из причин того, почему потенциал нейронных сетей в первые годы не был по-настоящему осознан.

Начало эпохи больших данных стало возможным благодаря прогрессу в технологиях сбора данных. В настоящее время практически вся информация о совершаемых нами действиях, включая покупку товаров, звонки по телефону или посещение сайтов, кем-то собирается и записывается. К тому же разработка мощных графических процессоров сделала возможным резкое повышение эффективности обработки огромных объемов данных. Эти достижения в значительной мере объясняют недавние успехи глубокого обучения, достигнутые с использованием алгоритмов, которые представляют собой лишь незначительно видоизмененные версии алгоритмов, доступных уже два десятилетия тому назад. Кроме того, эти недавние модификации алгоритмов стали возможными благодаря возросшей скорости вычислений, что создало благоприятные условия для тестирования моделей (и последующей корректировки алгоритмов). Если для тестирования алгоритма требуется месяц, то на одной аппаратной платформе за год может быть протестировано не более двенадцати его вариаций. Ситуации подобного рода исторически ограничивали интенсивность экспериментов, необходимых для настройки алгоритмов обучения нейронных сетей. Быстрые темпы прогресса во всем, что касается доступности данных, скорости вычислений и возможностей проведения экспериментов, усилили оптимизм в отношении будущих перспектив глубокого обучения. Ожидается, что к концу нынешнего столетия мощность компьютеров возрастет до такой степени, что станет возможным тренировать нейронные сети с количеством нейронов, сопоставимым с количеством нейронов в человеческом мозге. И хотя очень трудно предсказать истинные возможности нейронных сетей, которыми они будут обладать к тому времени, опыт разработок в области компьютерного зрения убеждает нас в том, что мы должны быть готовы к любым неожиданностям.

Структура главы

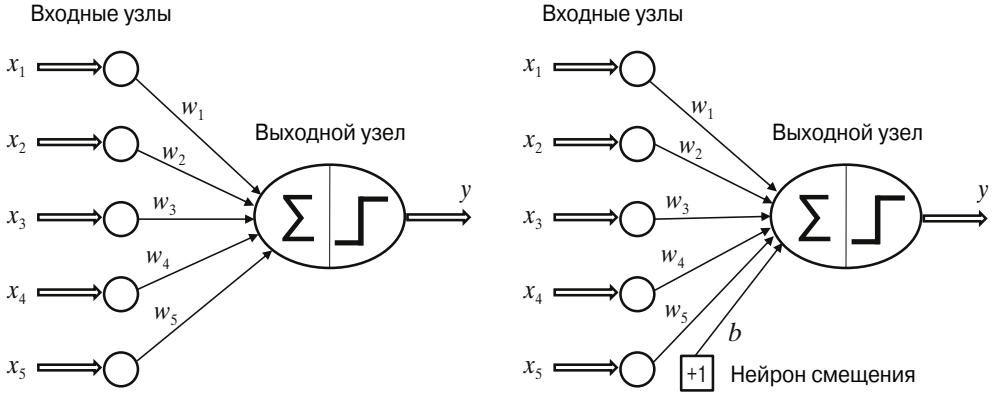
В следующем разделе вы познакомитесь с принципами работы одно- и многослойных сетей. Будут обсуждаться различные типы функций активации, выходных узлов и функций потерь. Рассмотрению алгоритма обратного распространения ошибки посвящен раздел 1.3. Практические аспекты тренировки нейронных сетей обсуждаются в разделе 1.4. Ключевые моменты, касающиеся того, каким образом нейронные сети обретают эффективность за счет выбора функций активации, описаны в разделе 1.5. Типичные архитектуры, используемые при проектировании нейронных сетей, обсуждаются в разделе 1.6. Дополнительные вопросы глубокого обучения рассмотрены в разделе 1.7, а некоторые показательные эталонные тесты — в разделе 1.8. Резюме главы приведено в разделе 1.9.

1.2. Базовая архитектура нейронных сетей

В этом разделе рассмотрены одно- и многослойные нейронные сети. В однослойной сети набор входов непосредственно транслируется на выход путем использования обобщенного варианта линейной функции. Этот простой вариант реализации нейронной сети также называют *перцептроном*. В многослойных нейронных сетях нейроны располагаются слоями, причем между входным и выходным слоями располагается группа так называемых *скрытых слоев*. Нейронные сети с такой многослойной архитектурой также называют *сетями прямого распространения* (feed-forward network). В этом разделе мы обсудим как однослойные, так и многослойные сети.

1.2.1. Одиночный вычислительный слой: перцептрон

Простейшая нейронная сеть, получившая название *перцептрон*, состоит из одного входного слоя и одного выходного. Базовая структура перцептрона приведена на рис. 1.3, *а*. Рассмотрим ситуацию, когда каждый тренировочный пример имеет вид $\left(\overline{X},\ y\right)$, где каждый вектор $\overline{X} = [x_1 \ldots x_d]$ содержит d переменных признаков, а $y \in \{-1,\ +1\}$ содержит наблюдаемое значение переменной бинарного класса. “Наблюдаемым” мы будем называть значение, предоставляемое нейронной сети в составе тренировочных данных, и наша цель заключается в том, чтобы научиться предсказывать значения переменной класса для случаев, в которых за ней не осуществлялось наблюдение. Например, в приложении, обнаруживающем поддельные кредитные карты, признаками могут служить различные свойства набора операций с кредитными картами (такие, как объемы и частота операций), а переменная класса может указывать на то, является ли



а) Перцептрон без смещения б) Перцептрон со смещением

Рис. 1.3. Базовая архитектура перцептрона

данный набор операций законным или незаконным. Ясно, что в приложениях такого типа необходимо располагать информацией о предыдущих случаях, в которых наблюдалась данная переменная класса, и о других (текущих) случаях, в которых переменная класса еще не наблюдалась, но должна быть предсказана.

Входной слой содержит d узлов, которые передают выходному узлу d признаков $\overline{X} = [x_1 \ldots x_d]$ с весовыми коэффициентами $\overline{W} = [w_1 \ldots w_d]$. Сам входной слой не выполняет сколь-нибудь существенных вычислений. Линейная функция $\overline{W} \cdot \overline{X} = \sum_{i=1}^{d} w_i x_i$ вычисляется на выходном узле. Впоследствии знак этого реального значения используется для предсказания значения зависимой переменной $\overline{X}$. Поэтому предсказанное значение $\hat{y}$ определяется следующей формулой:

$$\hat{y} = \text{sign}\{\overline{W} \cdot \overline{X}\} = \text{sign}\{\sum_{j=1}^{d} w_j x_j\}. \tag{1.1}$$

Функция *sign* переводит реальное значение в +1 или −1, что вполне подходит для бинарной классификации. Обратите внимание на символ циркумфлекса над переменной y, указывающий на то, что это предсказанное, а не наблюдаемое значение. Поэтому ошибка предсказания $E\left(\overline{X}\right) = y - \hat{y}$ может иметь одно из значений: $\{-2, 0, +2\}$. В тех случаях, когда ошибка $E\left(\overline{X}\right)$ имеет ненулевое значение, веса нейронной сети должны быть оптимизированы за счет обновления в направлении, противоположном направлению градиента ошибки. Как будет показано далее, этот процесс аналогичен тому, который используется в различных типах линейных моделей в машинном обучении. Несмотря на сходство перцептрона с традиционными моделями машинного обучения, его весьма полезно интерпретировать как вычислительный элемент, поскольку это позволяет объединять множество таких элементов для создания гораздо более мощных моделей, чем те, которые доступны в традиционном машинном обучении.

Архитектура перцептрона представлена на рис. 1.3, *а*, на котором одиночный входной слой передает признаки выходному узлу. Связи, соединяющие вход с выходом, характеризуются весами $w_1 \ldots w_d$, после умножения на которые признаки суммируются на выходном узле. Далее функция *sign* преобразует агрегированное значение в метку класса. Функция *sign* играет роль *функции активации*. Для имитации различных типов моделей, используемых в машинном обучении, можно выбирать различные функции активации, приводящие к таким моделям, как *регрессия по методу наименьших квадратов с числовыми целевыми значениями*, *метод опорных векторов* или *классификатор на основе логистической регрессии*. Большинство базовых моделей машинного обучения можно легко представить как различные архитектуры простой нейронной сети. Моделирование всевозможных методов традиционного машинного обучения как нейронных архитектур — занятие весьма полезное, поскольку это позволяет прояснить кар-

тину того, как глубокое обучение обобщает традиционное машинное обучение. Эта точка зрения детально исследуется в главе 2. Следует обратить внимание на то, что в действительности перцептрон содержит два слоя, хотя входной слой не выполняет никаких вычислений и лишь передает значения признаков. Поэтому при подсчете количества слоев в нейронной сети входной слой не учитывается. Поскольку перцептрон содержит только *один* вычислительный слой, он считается однослойной сетью.

Во многих случаях предсказание содержит инвариантную часть, так называемое *смещение* (bias). Рассмотрим, например, эксперимент, в котором переменные признаков центрированы на среднем значении, но среднее значение предсказания бинарных классов из набора {–1, +1} не равно нулю. Обычно такое встречается в ситуациях, в которых распределение бинарных классов в значительной степени разбалансировано. Тогда вышеупомянутого подхода оказывается недостаточно для выполнения предсказаний. В этом случае мы должны включать в рассмотрение дополнительную переменную смещения b, вбирающую в себя инвариантную часть предсказания:

$$\hat{y} = \text{sign}\{\overline{W} \cdot \overline{X} + b\} = \text{sign}\{\sum_{j=1}^{d} w_j x_j + b\}. \tag{1.2}$$

Смещение можно включить в качестве веса связи, используя *нейрон смещения*. Это достигается за счет добавления нейрона, который всегда передает значение 1 выходному узлу. Тогда вес связи, соединяющей нейрон смещения с выходным узлом, представляет собой переменную смещения. Пример нейрона смещения представлен на рис. 1.3, *б*. Другой подход, который хорошо работает в однослойных архитектурах, основан на *трюке с конструированием признаков*, предполагающем создание дополнительного признака с постоянным значением 1. Коэффициент при этом признаке представляет смещение, и тогда можно работать с уравнением 1.1. В книге мы не будем использовать смещения в явном виде, поскольку они могут быть учтены путем добавления нейронов смещения. При этом детали тренировочных алгоритмов остаются прежними за счет того, что мы рассматриваем нейрон смещения как любой другой нейрон с фиксированным значением активации, равным 1. Поэтому далее мы будем использовать для предсказаний уравнение 1.1, в котором смещения не фигурируют в явном виде.

В те времена, когда Розенблатт предложил алгоритм перцептрона [405], оптимизация весов осуществлялась эвристическими методами на аппаратном уровне и не представлялась в терминах формализованных задач оптимизации в машинном обучении (как это общепринято в наши дни). Однако конечной целью всегда ставилась минимизация ошибки предсказания, даже если эта задача и не формулировалась на формальном уровне. Поэтому эвристическая модель

перцептрона была спроектирована так, чтобы минимизировать число случаев ошибочной классификации, и были предложены способы доказательства сходимости процесса, гарантирующие корректность алгоритма обучения в упрощенных постановках экспериментов. Таким образом, мы все же можем сформулировать (обоснованную эвристически) цель алгоритма перцептрона в форме метода наименьших квадратов по отношению ко всем тренировочным примерам в наборе данных $\mathcal{D}$, содержащем пары "признак — метка":

$$\text{Минимизировать}_{\overline{W}} L = \sum_{(\overline{X},\, y)\in\mathcal{D})} (y - \hat{y})^2 = \sum_{(\overline{X},\, y)\in\mathcal{D})} (y - \text{sign}\{\overline{W}\cdot\overline{X}\})^2.$$

Целевые функции минимизации такого типа также называют *функциями потерь* (loss functions). Как будет показано далее, почти все алгоритмы обучения нейтральной сети формулируются с использованием понятия функции потерь. В главе 2 будет показано, что эта функция во многом подобна функции, используемой в регрессии по методу наименьших квадратов. Однако последняя определена для целевых переменных, имеющих непрерывные значения, и соответствующие потери являются гладкой и непрерывной функцией своих переменных. С другой стороны, в случае нашей целевой функции в форме метода наименьших квадратов функция *sign* не является дифференцируемой и имеет скачки в некоторых точках. К тому же она принимает постоянные значения на протяжении больших участков области изменения переменных и поэтому имеет нулевой градиент в тех точках, где она дифференцируема. В результате поверхность функции потерь имеет ступенчатую форму, непригодную для применения метода градиентного спуска. Алгоритм перцептрона использует (неявно) гладкую аппроксимацию градиента этой целевой функции по отношению к каждому примеру:

$$\nabla L_{\text{сглаженная}} = \sum_{(\overline{X},\, y)\in\mathcal{D})} (y - \hat{y})\overline{X}. \tag{1.3}$$

Обратите внимание на то, что вышеприведенный градиент не является истинным градиентом ступенчатой поверхности (эвристической) целевой функции, которая не предоставляет пригодных для использования градиентов. Поэтому "лестница" сглаживается и принимает форму наклонной поверхности, определяемой *критерием перцептрона*. Свойства критерия перцептрона будут описаны в разделе 1.2.1.1. Следует отметить, что такие понятия, как "критерий перцептрона", были предложены уже после выхода статьи Розенблатта [405] для объяснения эвристики шагов градиентного спуска. А пока что мы будем полагать, что алгоритм перцептрона оптимизирует некоторую неизвестную гладкую функцию с помощью градиентного спуска.

Несмотря на то что вышеприведенная целевая функция определена на всем диапазоне тренировочных данных, в процессе создания предсказания $\hat{y}$ обучающим алгоритмом нейронных сетей каждый пример входных данных $\overline{X}$ передается сети поочередно (или в составе небольших пакетов). После этого, исходя из значения ошибки $E\left(\overline{X}\right) = (y - \hat{y})$, выполняется обновление весов. В частности, при передаче сети точки данных $\overline{X}$ вектор весов $\overline{W}$ обновляется в соответствии с такой формулой:

$$\overline{W} \Leftarrow \overline{W} + \alpha(y - \hat{y})\overline{X}. \tag{1.4}$$

Параметр α регулирует *скорость обучения* (learning rate) нейронной сети. Алгоритм перцептрона повторяет циклические проходы по всем тренировочным примерам в случайном порядке и итеративно настраивает веса до тех пор, пока не будет достигнута сходимость. Прохождение одиночной точки данных может циклически повторяться множество раз. Каждый такой цикл называется *эпохой*. Формулу обновления весов по методу градиентного спуска можно переписать в терминах ошибки $E\left(\overline{X}\right) = (y - \hat{y})$ в следующем виде:

$$\overline{W} \Leftarrow \overline{W} + \alpha E\left(\overline{X}\right)\overline{X}. \tag{1.5}$$

Базовый алгоритм перцептрона можно рассматривать как метод *стохастического градиентного спуска* (stochastic gradient-descent), который неявно минимизирует квадрат ошибки предсказания, выполняя обновления методом градиентного спуска по отношению к случайно выбираемым тренировочным точкам. При этом суть основного допущения заключается в том, что нейронная сеть в процессе тренировки циклически обходит точки в случайном порядке и изменяет веса, стремясь уменьшить ошибку предсказания в данной точке. Как следует из уравнения 1.5, ненулевые обновления весов возможны только в том случае, если $y \neq \hat{y}$, т.е. только тогда, когда предсказание было выполнено с ошибкой. В методе *мини-пакетного стохастического градиентного спуска* вышеупомянутые обновления (1.5) реализуются на случайно выбранном поднаборе тренировочных точек S:

$$\overline{W} \Leftarrow \overline{W} + \alpha \sum_{\overline{X} \in S} E\left(\overline{X}\right)\overline{X}. \tag{1.6}$$

Преимущества использования мини-пакетного стохастического градиентного спуска обсуждаются в разделе 3.2.8. Интересной особенностью перцептрона является то, что скорость обучения α может быть установлена равной 1, поскольку она масштабирует лишь веса.

Модель, предложенная перцептроном, относится к *линейному типу*, для которого уравнение $\overline{W} \cdot \overline{X} = 0$ определяет линейную гиперплоскость. Здесь $\overline{W} = (w_1 \ldots w_d)$ — d-мерный вектор, направленный вдоль нормали к гиперплоскости.

Кроме того, значение $\overline{W} \cdot \overline{X}$ положительно для значений $\overline{X}$ по одну сторону от гиперплоскости и отрицательно для значений $\overline{X}$ по другую ее сторону. Такой тип моделей особенно хорошо работает в случае *линейно разделимых* данных. Примеры линейно разделимых и линейно неразделимых данных приведены на рис. 1.4.

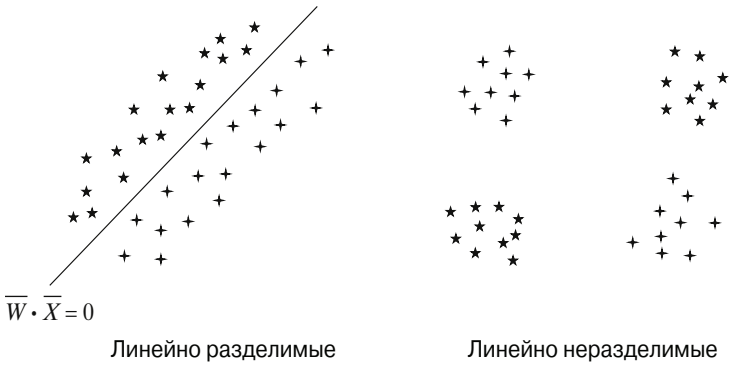


Рис. 1.4. Примеры линейно разделимых и неразделимых данных двух классов

Алгоритм перцептрона хорошо работает при классификации наборов данных, подобных тому, который представлен на рис. 1.4, *слева*, когда данные поддаются *линейному разделению*. С другой стороны, он плохо справляется с этой задачей в случае наборов данных вроде того, который представлен на рис. 1.4, *справа*. Этот пример демонстрирует присущие перцептрону ограничения, что заставляет прибегать к более сложным нейронным архитектурам.

Поскольку оригинальный механизм перцептрона был предложен в качестве метода эвристической минимизации ошибок классификации, было особенно важно показать, что данный алгоритм сходится к разумным решениям в некоторых специальных случаях. В этом контексте было доказано [405], что условием сходимости алгоритма перцептрона к нулевой ошибке на тренировочных данных является линейная разделимость этих данных. Однако алгоритм перцептрона не гарантирует сходимость для примеров, в которых данные не являются линейно разделимыми. По причинам, которые обсуждаются в следующем разделе, в случае данных, не являющихся линейно разделимыми, алгоритм перцептрона иногда может приводить к очень плохим решениям (по сравнению со многими другими алгоритмами обучения).

1.2.1.1. Какой целевой функции соответствует оптимизация перцептрона

Как уже упоминалось, в оригинальной статье Розенблатта, в которой он описал перцептрон [405], функция потерь формально не вводилась. В те годы модели реализовывались аппаратными средствами. Оригинальный *перцептрон Mark I*, реализованный на компьютере, название которого и дало ему имя, изначально задумывался как машина, а не как алгоритм, в связи с чем для него было специально создано соответствующее электронное устройство (рис. 1.5).

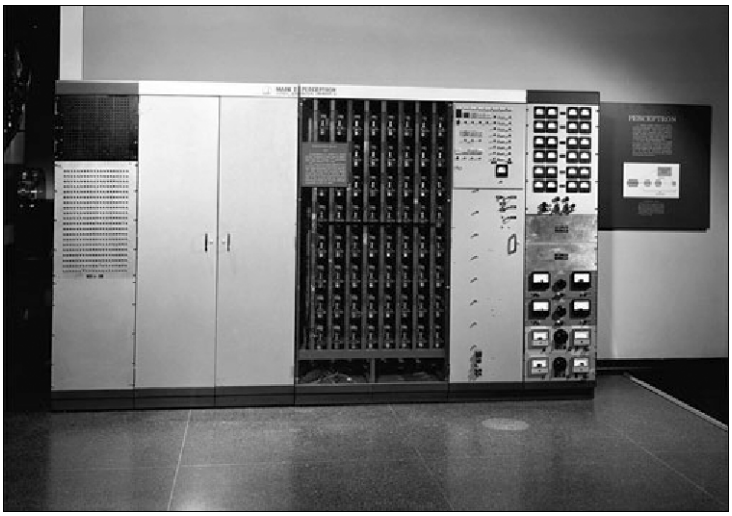


Рис. 1.5. Алгоритм перцептрона первоначально был реализован в виде электронного устройства; на приведенной фотографии изображен компьютер Mark I, созданный в 1958 году специально для реализации перцептрона (публикуется с разрешения Смитсоновского института)

Общая задача заключалась в минимизации количества ошибок классификации с помощью эвристического процесса обновления (реализованного аппаратными средствами), который изменялся в “правильном” направлении, если возникала ошибка. Это эвристическое обновление очень напоминало градиентный спуск, но оно получалось не по методу градиентного спуска. В алгоритмической постановке задачи градиентный спуск определен лишь для гладких функций потерь, тогда как аппаратный подход основывался на эвристических принципах и был рассчитан на *бинарные выходы*. Многие из принципов построения аппаратных моделей с бинарными выходами были заимствованы из *модели нейрона Маккаллока — Питтса* [321]. К сожалению, оптимизация по непрерывным переменным неприменима к бинарным сигналам.

Можем ли мы найти такую гладкую функцию потерь, чтобы ее градиент описывал обновление перцептрона? Количество ошибок классификации для бинарной задачи может быть записано в форме *двоичной функции потерь* (0/1 loss function) для тренировочной точки данных $\left(\overline{X}_i,\ y_i\right)$ в следующем виде:

$$L_i^{(0/1)} = \frac{1}{2}\left(y_i - \text{sign}\{\overline{W} \cdot \overline{X}_i\}\right)^2 = 1 - y_i \cdot \text{sign}\{\overline{W} \cdot \overline{X}_i\}. \tag{1.7}$$

Здесь правая часть целевой функции упрощена за счет замены значением 1 как члена y_i^2, так и члена $\text{sign}\{\overline{W} \cdot \overline{X}_i\}^2$, поскольку они получаются возведением в квадрат значения, извлекаемого из множества $\{-1, +1\}$. Однако эта целевая функция не является дифференцируемой, поскольку она имеет ступенчатую форму, особенно после ее суммирования по многим точкам. Обратите внимание на то, что в двоичной функции потерь доминирует член $-y_i \cdot \text{sign}\{\overline{W} \cdot \overline{X}_i\}$, в котором функция *sign* является источником большинства проблем, связанных с недифференцируемостью. Поскольку нейронные сети определяются оптимизацией на основе градиента, мы должны определить гладкую целевую функцию, которая отвечала бы за обновления перцептрона. Можно показать [41], что обновления перцептрона неявно оптимизируют *критерий перцептрона*. Эта целевая функция получается опусканием функции *sign* в приведенном выше выражении для двоичной функции потерь и подстановкой нулевого значения вместо отрицательных значений, чтобы все корректные значения обрабатывались унифицированным способом и без потерь:

$$L_i = \max\{-y_i(\overline{W} \cdot \overline{X}_i),\ 0\}. \tag{1.8}$$

Читателям предлагается самостоятельно использовать дифференциальное исчисление для проверки того, что градиент этой сглаженной целевой функции приводит к обновлению перцептрона, и это обновление по сути описывается формулой $\overline{W} \Leftarrow \overline{W} - \alpha \nabla_W L_i$. Видоизмененную функцию потерь, обеспечивающую возможность вычисления градиента недифференцируемой функции, также называют *сглаженной суррогатной функцией потерь* (smoothed surrogate loss function). Подобные функции используются почти всеми непрерывными методами обучения на основе оптимизации с дискретными выходами (такими, как метки классов).

Несмотря на то что вышеупомянутый критерий перцептрона был получен методами обратного инжиниринга, т.е. реконструирован по обновлениям перцептрона, природа этой функции потерь обнаруживает некоторые слабые стороны обновлений в оригинальном алгоритме. Критерий перцептрона имеет интересную особенность: можно установить $\overline{W}$ равным нулевому вектору, *независимо от тренировочного набора*, и получить оптимальное значение потерь, равное нулю. Несмотря на это, обновления сходятся к отчетливому разделителю между двумя классами в случае их линейной разделимости, и в конце

концов разделитель классов также обеспечивает нулевое значение потерь. Однако в случае данных, не разделяемых линейно, такое поведение становится достаточно произвольным, а иногда результирующее решение вообще не является хорошим приближением к разделителю классов. Непосредственная чувствительность потерь к *величине* вектора весов может размыть цель разделения классов. Возможны даже ситуации, когда количество случаев ошибочной классификации значительно увеличивается, хотя потери при этом уменьшаются. Это является ярким примером того, как суррогатные функции потерь иногда не в состоянии обеспечить решение поставленных перед ними задач. В силу указанных причин описанный подход отличается нестабильностью и может приводить к решениям, качество которых меняется в широких пределах.

В связи с этим были предложены различные варианты алгоритма обучения для неразделимых данных, при этом естественный подход состоит в том, чтобы всегда отслеживать, какое решение является наилучшим в смысле наименьшего количества случаев ошибочной классификации [128]. Такой подход, когда наилучшее решение всегда держится наготове, получил название *карманного алгоритма* (pocket algorithm). Другой в высшей степени эффективный вариант, включающий понятие *зазоров* для функции потерь, приводит к созданию алгоритма, *идентичного линейному методу опорных векторов* (Support Vector Machine — SVM). По этой причине линейный метод опорных векторов также называют *перцептроном оптимальной стабильности*.

1.2.1.2. Взаимосвязь с методом опорных векторов

Критерий перцептрона — это смещенная версия функции *кусочно-линейных потерь* (hinge-loss), используемой в методе опорных векторов (глава 2). Кусочно-линейные потери еще более похожи на двоичный критерий потерь (см. уравнение 1.7) и определяются следующей формулой:

$$L_i^{svm} = \max\{1 - y_i(\overline{W} \cdot \overline{X}_i),\ 0\}. \tag{1.9}$$

Обратите внимание на отсутствие постоянного члена 1 в правой части варианта уравнения 1.7, соответствующего перцептрону, тогда как в случае кусочно-линейных потерь данная константа включается в максимизируемую функцию. Это изменение не сказывается на алгебраическом выражении для градиента, но влияет на принятие решений о том, какие точки не вносят вклад в потери и не должны обновляться. Связь между критерием перцептрона и кусочно-линейными потерями показана на рис. 1.6. Это сходство становится особенно очевидным, если переписать обновления перцептрона, определяемые уравнением 1.6, в следующем виде:

$$\overline{W} \Leftarrow \overline{W} + \alpha \sum_{(\overline{X},\, y) \in S^+} y\overline{X}. \tag{1.10}$$

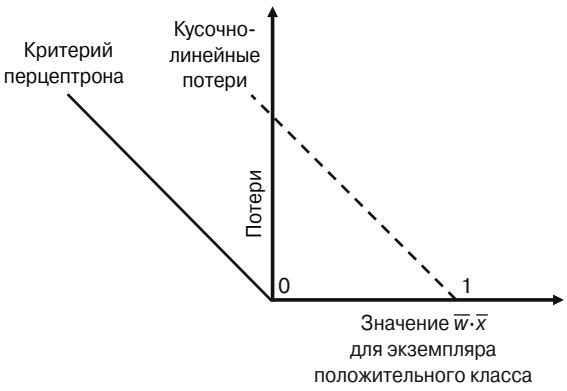


Рис. 1.6. Критерий перцептрона и кусочно-линейные потери

Здесь S^+ определен как набор всех неправильно классифицированных тренировочных точек $\overline{X} \in S$, которые удовлетворяют условию $y(\overline{W} \cdot \overline{X}) < 0$. Очевидно, что это обновление несколько отличается от обновления для перцептрона, поскольку перцептрон использует для обновления ошибку $E(\overline{X})$, которая в вышеприведенном уравнении заменяется на y. Здесь ключевым является то обстоятельство, что (целочисленная) ошибка $E(\overline{X}) = (y - \text{sign}\{\overline{W} \cdot \overline{X}\}) \in \{-2, +2\}$ никогда не может иметь нулевое значение для ошибочно классифицированных точек в S^+. Поэтому для *ошибочно классифицированных точек* $E(\overline{X}) = 2y$, и тогда, предварительно включив множитель 2 в скорость обучения, $E(\overline{X})$ в обновлениях можно заменить на y. Это обновление идентично тому, которое применялось в первоначальном алгоритме метода опорных векторов (SVM) [448], за исключением того, что в перцептроне обновления вычисляются только для ошибочно классифицированных точек, тогда как SVM дополнительно учитывает *зазоры* для корректных точек вблизи границ принятия решений относительно обновлений. Обратите внимание на то, что SVM использует для определения S^+ условие $y(\overline{W} \cdot \overline{X}) < 1$ (вместо $y(\overline{W} \cdot \overline{X}) < 0$) — одно из основных различий между этими двумя алгоритмами. Отсюда следует, что перцептрон по сути не слишком отличается от таких хорошо известных алгоритмов, как метод опорных векторов, хотя в его основе и лежат другие предпосылки. Фройнд и Шапир предоставили отличное объяснение роли зазоров в улучшении стабильности перцептрона и его связи с методом опорных векторов [123]. Получается, что многие традиционные модели машинного обучения можно рассматривать как незначительные вариации мелких нейронных архитектур, таких как перцептрон. Связь между классическими моделями машинного обучения и мелкими нейронными сетями подробно описана в главе 2.

1.2.1.3. Выбор функции активации и функции потерь

Выбор функции активации — критическая часть процесса проектирования нейронной сети. В случае перцептрона выбор функции *sign* в качестве функции активации обусловлен тем фактом, что предсказания касаются бинарных меток классов. Однако возможны ситуации другого типа, требующие предсказания других целевых переменных. Если, скажем, предсказываемая целевая переменная имеет вещественные значения, то имеет смысл использовать тождественную функцию активации, и в этом случае результирующий алгоритм совпадает с алгоритмом регрессии по методу наименьших квадратов. Если желательно предсказывать вероятность бинарного класса, то для активации выходного узла целесообразно использовать *сигмоиду*, чтобы прогнозное значение $\hat{y}$ указывало на вероятность того, что наблюдаемое значение y зависимой переменной равно 1. Исходя из предположения, что y кодируется из множества значений $\{-1, 1\}$, в качестве функции потерь используется отрицательный логарифм $\left|y/2-0{,}5+\hat{y}\right|$. Если $\hat{y}$ — это вероятность того, что y равен 1, то $\left|y/2-0{,}5+\hat{y}\right|$ — это вероятность предсказания корректного значения. В том, что это действительно так, можно легко убедиться, проверив два случая, когда y равно 0 или 1. Можно показать, что эта функция потерь представляет отрицательное логарифмическое правдоподобие тренировочных данных (раздел 2.2.3). Нелинейность функций активации приобретает особое значение при переходе от однослойного перцептрона к многослойным архитектурам, о которых речь пойдет далее. В различных слоях могут использоваться различные типы нелинейных функций, такие как *знаковая* (сигнум), *сигмоида* и *гиперболический тангенс*. Мы будем использовать для функции активации обозначение Ф:

$$\hat{y} = \Phi(\overline{W} \cdot \overline{X}). \tag{1.11}$$

Таким образом, в действительности нейрон вычисляет в пределах узла две функции, и именно поэтому наряду с символом суммирования S мы включаем в нейрон символ функции активации Ф. Разбиение выполняемых нейроном вычислений на два отдельных значения показано на рис. 1.7.

По отношению к значениям, вычисляемым до применения к ним функции активации Ф(·), мы будем использовать термин *предактивационное значение*, тогда как по отношению к значениям, вычисленным после применения функции активации, — термин *постактивационное значение*. Выходом нейрона всегда является постактивационное значение, хотя предактивационные переменные также часто используются в различных типах анализа, таких, например, как алгоритм *обратного распространения ошибки* (backpropagation algorithm), к обсуждению которого мы еще вернемся далее. Происхождение пред- и постактивационных значений показано на рис. 1.7.

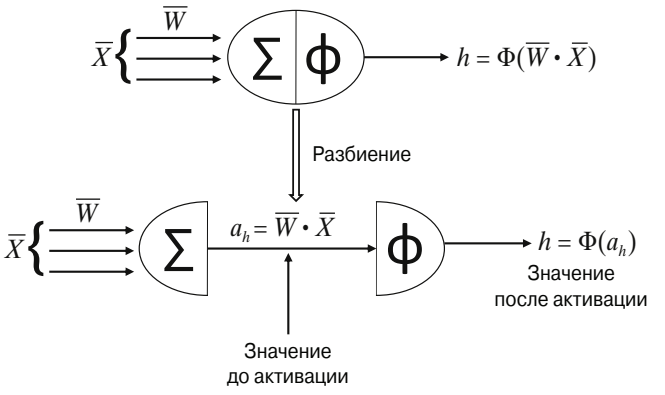


Рис. 1.7. Значения, вычисляемые в нейроне до и после активации

Простейшей функцией активации $\Phi(\cdot)$ является тождественная или линейная функция, вообще не вносящая никакой нелинейности:

$$\Phi(v) = v.$$

Линейная активационная функция часто используется в выходных узлах в тех случаях, когда целевое значение является вещественным числом. Ее используют даже для дискретных выходов, если необходимо задать сглаженную суррогатную функцию потерь.

К числу классических функций активации, которые применялись на ранних этапах разработки нейронных сетей, относятся знаковая функция, или сигнум (*sign*), сигмоида (*sigmoid*) и гиперболический тангенс (*tanh*):

$$\Phi(v) = \text{sign}(v) \quad (\text{сигнум}),$$

$$\Phi(v) = \frac{1}{1+e^{-v}} \quad (\text{сигмоида}),$$

$$\Phi(v) = \frac{e^{-2v}-1}{e^{-2v}+1} \quad (\text{гиперболический тангенс}).$$

Несмотря на то что функцию активации *sign* можно применять для трансляции значений на бинарные выходы на этапе предсказания, тот факт, что она не является дифференцируемой функцией, препятствует ее использованию для создания функции потерь на этапе тренировки. Например, перцептрон задействует функцию *sign* для предсказаний, тогда как критерий перцептрона требует использовать во время тренировки только линейную активацию. Выходное значение сигмоиды принадлежит к интервалу [0, 1], что удобно в тех случаях, когда результаты вычислений должны интерпретироваться как вероятностные значения. Это также оказывается полезным при создании вероятностных выходов и конструировании функций потерь, получаемых из моделей

максимального правдоподобия. Гиперболический тангенс имеет примерно ту же форму, что и сигмоида, за исключением того, что функция *tanh* масштабирована вдоль горизонтальной оси, а также масштабирована и сдвинута вдоль вертикальной оси таким образом, что ее значения принадлежат к интервалу [−1, 1]. Связь между гиперболическим тангенсом и сигмоидой описывается следующей формулой (см. упражнение 3):

$$\tanh(v) = 2 \cdot \text{sigmoid}(2v) - 1.$$

Гиперболический тангенс более предпочтителен, чем сигмоида, в тех случаях, когда желательно, чтобы результаты вычислений могли иметь как положительные, так и отрицательные значения. Кроме того, его центрированность на среднем значении и больший (из-за более сжатой формы функции) градиент по сравнению с сигмоидой упрощают тренировку модели. Ранее для введения нелинейности в нейронные сети в качестве функций активации традиционно использовали сигмоиду или гиперболический тангенс. Однако в последние годы все бо́льшую популярность приобретают различные кусочно-линейные функции активации наподобие тех, которые приведены ниже.

$\Phi(v) = \max\{v,\ 0\}$ (полулинейный элемент [ReLU]),

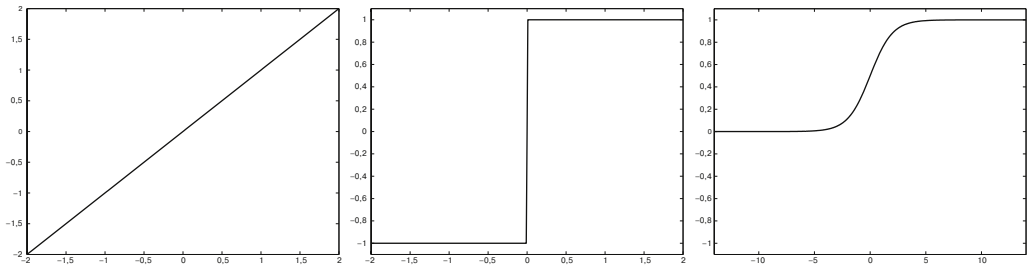
$\Phi(v) = \max\{\min[v,\ 1],\ -1\}$ (спрямленный гиперболический тангенс).

В современных нейронных сетях функции активации ReLU (Rectified Linear Unit) и спрямленный гиперболический тангенс в значительной степени вытеснили сигмоиду и гиперболический тангенс, поскольку их использование упрощает тренировку многослойных нейронных сетей.

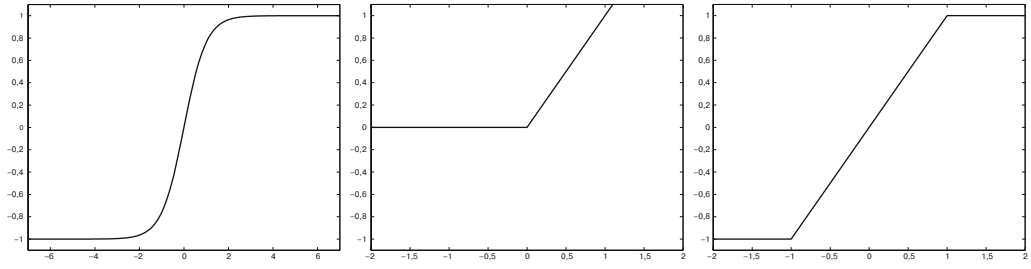
Вышеупомянутые функции активации представлены на рис. 1.8. Заслуживает внимания тот факт, что все они являются монотонными функциями. Кроме того, за исключением тождественной функции, в своем большинстве[1] они *насыщаются* при больших абсолютных значениях аргумента, т.е. не изменяются существенно при дальнейшем увеличении его абсолютного значения.

Как будет показано далее, подобные нелинейные функции активации находят широкое применение в многослойных нейронных сетях, поскольку они помогают создавать более мощные композиции функций различного типа. Многие из этих функций называют функциями *сжатия*, поскольку они транслируют выходные значения из произвольного интервала на ограниченный интервал значений. Использование нелинейной активации играет фундаментальную роль в увеличении моделирующей способности сети. Если сеть использует только линейные виды активации, то она не сможет выйти за пределы возможностей однослойной нейронной сети. Эта проблема обсуждается в разделе 1.5.

[1] Функция активации ReLU демонстрирует асимметричное насыщение.



а) Тождественная б) Знаковая в) Сигмоида



г) Гиперболический тангенс д) ReLU е) Спрямленный гиперболический тангенс

Рис. 1.8. Разновидности функции активации

1.2.1.4. Выбор выходных узлов и их количества

Выбор выходных узлов и их количества в значительной степени зависит от выбора функции активации, что, в свою очередь, определяется конкретной задачей. Например, в случае *k*-арной классификации могут использоваться *k* выходных значений $\overline{v} = [v_1 \ldots v_k]$ с применением функции активации *Softmax* в узлах данного слоя. В частности, функция активации для выхода определяется следующим образом:

$$\Phi\left(\overline{v}\right)_i = \frac{\exp(v_i)}{\sum_{j=1}^{k} \exp(v_j)} \quad \forall i \in \{1 \ldots k\}. \tag{1.12}$$

Эти *k* значений полезно рассматривать как выходные значения *k* узлов, входными значениями которых являются $v_1 \ldots v_k$. Пример функции *Softmax* с тремя выходами приведен на рис. 1.9, на котором также представлены соответствующие значения v_1, v_2 и v_3. Обратите внимание на то, что три выхода соответствуют трем классам, и они преобразуют выходные значения последнего скрытого слоя в вероятности с помощью функции *Softmax*. В последнем скрытом слое часто используют линейную (тождественную) функцию активации, если его

выход передается в качестве входа слою Softmax. Кроме того, со слоем Softmax не связаны никакие веса, поскольку он лишь преобразует вещественные значения в вероятности. Использование функции активации *Softmax* с единственным скрытым слоем линейной активации в точности реализует модель так называемой *мультиномиальной логистической регрессии* (multinomial logistic regression) [6]. Аналогичным образом с помощью нейронных сетей могут быть легко реализованы многие вариации методов наподобие мультиклассовых SVM. В качестве еще одного типичного примера использования множественных выходных узлов можно привести *автокодировщик*, в котором каждая входная точка данных полностью реконструируется выходным слоем. Автокодировщики могут применяться для реализации методов матричной факторизации наподобие *сингулярного разложения* (singular value decomposition). Эта архитектура будет подробно обсуждаться в главе 2. Изучение простейших нейронных сетей, имитирующих базовые алгоритмы машинного обучения, может быть весьма поучительным, поскольку они занимают промежуточное положение между традиционным машинным обучением и глубокими сетями. Исследование этих архитектур позволяет лучше понять связь между традиционным машинным обучением и нейронными сетями, а также преимущества, обеспечиваемые последними.

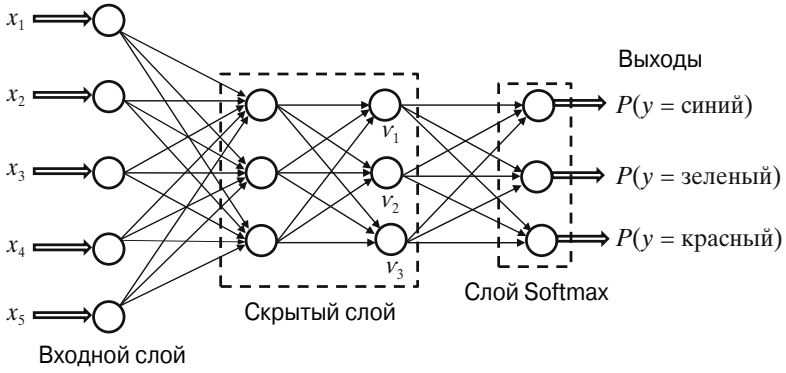


Рис. 1.9. Пример множественных выходов для категориальной классификации с использованием слоя Softmax

1.2.1.5. Выбор функции потерь

При определении выходов, наиболее подходящих для данной задачи, очень многое зависит от правильного выбора функции потерь. Например, регрессия по методу наименьших квадратов с числовыми выходами требует использования простой квадратичной функции потерь вида $(y - \hat{y})^2$ для простого тренировочного примера с целевым значением y и предсказанием $\hat{y}$. Можно задейство-

вать и другие типы потерь наподобие *кусочно-линейной функции* $y \in \{-1, +1\}$ и вещественного предсказания $\hat{y}$ (с тождественной функцией активации):

$$L = \max\{0, 1 - y \cdot \hat{y}\}. \tag{1.13}$$

Кусочно-линейные потери можно применять для реализации метода обучения, известного как *метод опорных векторов* (Support Vector Machine — SVM).

Для множественных предсказаний (таких, как предсказание идентификаторов слов или одного из нескольких классов) особенно полезен выход Softmax. Однако выход такого типа — вероятностный и поэтому требует использования функции потерь другого типа. В действительности для вероятностных предсказаний используют два различных типа функций потерь, в зависимости от того, является ли предсказание бинарным или многоарным.

1. **Бинарные целевые значения (логистическая регрессия).** В этом случае предполагается, что наблюдаемое значение y извлекается из набора $\{-1, +1\}$, а предсказание $\hat{y}$ является произвольным числовым значением при использовании тождественной функции активации. В подобных ситуациях функция потерь для одиночного примера с наблюдаемым значением y и предсказанием $\hat{y}$ в виде вещественного числа (с тождественной активацией) определяется как

$$L = \log\left(1 + exp\left(-y \cdot \hat{y}\right)\right). \tag{1.14}$$

   Функция потерь этого типа реализует фундаментальный метод машинного обучения, известный как *логистическая регрессия*. Другой возможный подход заключается в использовании сигмоиды для получения выхода $y \in \{0, 1\}$, определяющего вероятность того, что наблюдаемое значение y равно 1. Далее, отрицательный логарифм $\left|y / 2 - 0{,}5 + \overline{y}\right|$ представляет потери в предположении, что y кодируется из набора $\{-1, 1\}$. Это обусловлено тем, что $\left|y / 2 - 0{,}5 + \overline{y}\right|$ определяет вероятность того, что предсказание корректно. Данное наблюдение иллюстрирует тот факт, что для достижения одного и того же результата можно использовать различные комбинации функции активации и функции потерь.

2. **Категориальные целевые значения.** В данном случае, если $\hat{y}_1 \ldots \hat{y}_k$ — вероятности k классов (в уравнении 1.9 используется функция активации *Softmax*) и r-й класс — истинный, то функция потерь для одиночного примера определяется следующим образом:

$$L = -\log\left(\hat{y}_r\right). \tag{1.15}$$

   Функция потерь этого типа реализует мультиномиальную логистическую регрессию и известна под названием *кросс-энтропийные потери*

(cross-entropy loss). Заметьте, что бинарная логистическая регрессия идентична мультиномиальной логистической регрессии при $k = 2$.

Очень важно не забывать о том, что выбор выходных узлов, функции активации и функции потерь определяется конкретной задачей. Кроме того, эти варианты выбора также взаимозависимы. Несмотря на то что перцептрон часто рассматривают как квинтэссенцию однослойных сетей, он представляет лишь одну из многочисленных возможностей. На практике критерий перцептрона редко используется в качестве функции потерь. В случае дискретных выходных значений обычно используют активацию *Softmax* с кросс-энтропийными потерями, в случае вещественных — линейную активацию с квадратичными потерями. Как правило, оптимизировать кросс-энтропийные потери легче, чем квадратичные.

1.2.1.6. Некоторые полезные производные функций активации

В большинстве случаев обучение нейронных сетей главным образом связано с градиентным спуском и функциями активации. По этой причине производные функций активации многократно используются в книге, и их обсуждение в одном месте будет полезным. В этом разделе подробно описаны производные функций потерь. В последующих главах мы будем неоднократно обращаться к приведенным здесь результатам.

1. *Линейная и знаковая функции активации.* Производная линейной функции активации везде равна 1. Производная функции $sign(v)$ равна 0 при всех значениях v, отличных от точки $v = 0$, где она разрывна и не имеет производной. Из-за нулевого значения градиента этой функции и ее недифференцируемости ее редко применяют в функции потерь, даже если она и используется для получения предсказаний на этапе тестирования. Производные линейной и знаковой функций активации представлены в графическом виде на рис. 1.10, *а* и *б*, соответственно.
2. *Сигмоида.* Производная сигмоиды выглядит особенно просто, будучи выраженной в терминах выхода сигмоиды, а не входа. Обозначим через o выход сигмоиды с аргументом v:

$$o = \frac{1}{1 + \exp(-v)}. \tag{1.16}$$

Тогда производную функции активации можно записать в следующем виде:

$$\frac{\partial o}{\partial v} = \frac{\exp(-v)}{(1 + \exp(-v))^2}. \tag{1.17}$$

Очень важно, что данное выражение можно переписать в более удобной форме в терминах выходов:

$$\frac{\partial o}{\partial v} = o(1-o). \tag{1.18}$$

Производную сигмоиды часто используют в виде функции выхода, а не входа. График производной сигмоиды приведен на рис. 1.10, *в*.

3. *Функция активации tanh.* Как и в случае сигмоиды, активацию *tanh* часто используют в качестве функции выхода o, а не входа v:

$$o = \frac{\exp(2v)-1}{\exp(2v)+1}. \tag{1.19}$$

Тогда градиент этой функции можно выразить в следующем виде:

$$\frac{\partial o}{\partial v} = \frac{4\cdot\exp(2v)}{(\exp(2v)+1)^2}. \tag{1.20}$$

Это выражение можно переписать в терминах выхода o:

$$\frac{\partial o}{\partial v} = 1-o^2. \tag{1.21}$$

График производной функции активации в виде гиперболического тангенса приведен на рис. 1.10, *г*.

4. *Функции активации ReLU и спрямленный гиперболический тангенс*. Производная функции активации ReLU равна 1 для неотрицательных значений аргумента и нулю в остальных случаях. Производная спрямленного гиперболического тангенса равна 1 для значений аргумента в интервале [−1, +1] и нулю в остальных случаях. Графики производных функции ReLU и спрямленного гиперболического тангенса приведены на рис. 1.10, *д* и *е*, соответственно.

1.2.2. Многослойные нейронные сети

Перцептрон содержит входной и выходной слои, из которых вычисления выполняются только в выходном слое. Входной слой передает данные выходному, и все вычисления полностью видны пользователю. Многослойные нейронные сети содержат несколько вычислительных слоев. Дополнительные промежуточные слои (расположенные между входом и выходом) называют *скрытыми слоями*, поскольку выполняемые ими вычисления остаются невидимыми для пользователя. Базовую архитектуру многослойных нейронных сетей называют *сетями прямого распространения* (feed-forward networks), поскольку в сетях этого типа данные последовательно передаются от одного слоя к другому в прямом направлении от входа к выходу. В используемой по умолчанию архитектуре

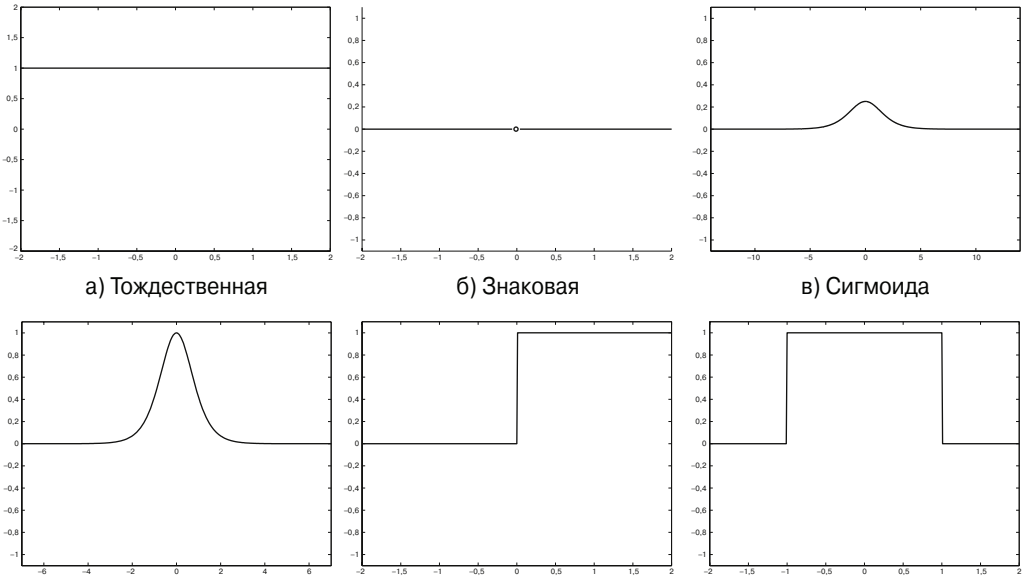
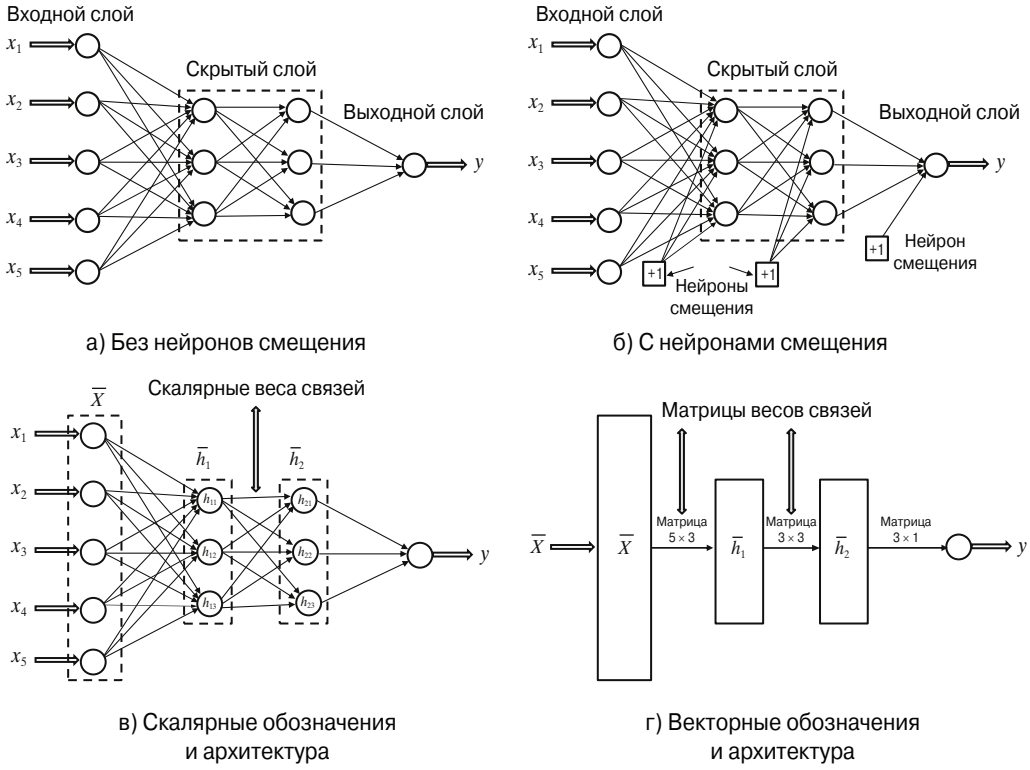


Рис. 1.10. Производные различных функций активации

сетей прямого распространения предполагается, что каждый узел одного слоя связан с каждым узлом следующего слоя. Поэтому архитектура такой сети почти полностью определяется количеством слоев и количеством/типом узлов в каждом слое. Единственное, что остается дополнительно определить, — это функцию потерь, которая оптимизируется в выходном слое. И хотя в алгоритме перцептрона используется критерий перцептрона, это не единственный вариант выбора. Очень часто используют выходы Softmax с кросс-энтропийной функцией потерь для предсказания дискретных значений и линейные выходы с квадратичной функцией потерь для предсказания вещественных значений.

Как и в случае однослойных сетей, в скрытых и выходных слоях многослойных сетей можно использовать нейроны смещения. Примеры многослойных нейронных сетей с нейронами смещения и без них приведены на рис. 1.11, *а* и *б*, соответственно. В каждом из этих случаев нейронная сеть содержит три слоя. Обратите внимание на то, что входной слой часто не учитывают при подсчете общего количества слоев, поскольку он просто передает данные следующему слою, не выполняя самостоятельно никаких вычислений. Если нейронная сеть содержит элементы $p_1 \dots p_k$ в каждом из своих k слоев, то представления их выходов $\hat{h}_1 \dots \hat{h}_k$ в виде векторов (столбцов) имеют размерности $p_1 \dots p_k$. Поэтому количество элементов в каждом слое называют *размерностью* (*размером*) данного слоя.



а) Без нейронов смещения

б) С нейронами смещения

в) Скалярные обозначения и архитектура

г) Векторные обозначения и архитектура

Рис. 1.11. Базовая архитектура сети прямого распространения с двумя скрытыми слоями и одним выходным слоем. Несмотря на то что каждый элемент содержит одну скалярную переменную, всю совокупность элементов одного слоя часто представляют в виде одного векторного элемента. Векторные элементы часто изображают в виде прямоугольников с указанием матриц весов связей на соединениях между ними

Веса связей, соединяющих входной слой с первым скрытым слоем, содержатся в *матрице* W_1 размера $d \times p_1$, тогда как веса связей между r-м и $(r+1)$-м скрытыми слоями — в матрице W_r размера $p_r \times p_{r+1}$. Если выходной слой содержит o узлов, то заключительная матрица W_{k+1} имеет размер $p_k \times o$. d-мерный входной вектор x преобразуется в выходы с использованием следующих рекурсивных уравнений:

$$\begin{aligned}
\overline{h}_1 &= \Phi(W_1^T \overline{x}) && \text{[входной со скрытыми]},\\
\overline{h}_{p+1} &= \Phi(W_{p+1}^T \overline{h}_p) \;\; \forall p \in \{1 \ldots k-1\} && \text{[скрытый со скрытыми]},\\
\overline{o} &= \Phi(W_{k+1}^T \overline{h}_k) && \text{[скрытый с выходным]}.
\end{aligned}$$

В этих уравнениях функции активации наподобие сигмоиды применяются к своим векторным аргументам *поэлементно*. Однако для некоторых функций активации, таких как *Softmax* (которые, как правило, используются в выходных

слоях), естественными являются векторные аргументы. И хотя каждый элемент нейронной сети содержит одну переменную, на многих архитектурных диаграммах эти элементы объединяются в один слой для создания единого векторного элемента, обозначаемого *прямоугольником*, а не *кружочком*. Так, архитектурная диаграмма, приведенная на рис. 1.11, *в* (со скалярными элементами), преобразована в векторную нейронную архитектуру, приведенную на рис. 1.11, *г*. Обратите внимание на то, что теперь соединения между векторными элементами представлены матрицами. Кроме того, неявно предполагается, что в векторных нейронных архитектурах во всех элементах данного слоя используется одна и та же функция активации, применяемая поэлементно ко всему слою. Обычно это ограничение не является проблемой, поскольку в большинстве нейронных архитектур вдоль всей цепочки вычислений используется одна и та же функция активации, с единственным исключением из этого правила, обусловленным природой выходного слоя. На протяжении всей книги элементы нейронных архитектур, содержащие векторные переменные, будут изображаться в виде прямоугольников, тогда как скалярным переменным будут соответствовать кружочки.

Имейте в виду, что вышеприведенные рекуррентные уравнения и векторные архитектуры справедливы лишь для сетей прямого распространения, структура которых представлена слоями, и не всегда могут использоваться в случае нестандартных архитектурных решений. К таким нестандартным решениям относятся любые архитектуры с входами, включенными в промежуточные слои, или топологиями, допускающими соединения между несмежными слоями. Кроме того, функции, вычисляемые в узле, не всегда могут представлять собой некоторую комбинацию линейной функции и функции активации. Для выполнения вычислений в узлах могут использоваться любые функции.

На рис. 1.11 был представлен строго классический тип архитектур сетей, однако он может варьироваться самыми разными способами, такими, например, как введение нескольких выходных слоев. Возможные варианты выбора часто диктуются условиями конкретной задачи (такой, например, как классификация или снижение размерности). Классическим примером задачи снижения размерности является автокодировщик, цель которого — воссоздание выходов на основании входов. В этом случае количество выходов совпадает с количеством входов (рис. 1.12). Расположенный посередине скрытый слой выводит уменьшенное представление каждого примера. В результате этого ограничения часть информации теряется, что обычно сопровождается появлением шума в данных. Выходы скрытых слоев соответствуют представлению данных, имеющему меньшую размерность. В действительности можно показать, что мелкий вариант этой схемы с математической точки зрения эквивалентен такому хорошо известному методу снижения размерности данных, как *сингулярное разложение*.

Как вы узнаете из главы 2, увеличение глубины сети позволяет еще существеннее снижать размерность данных.

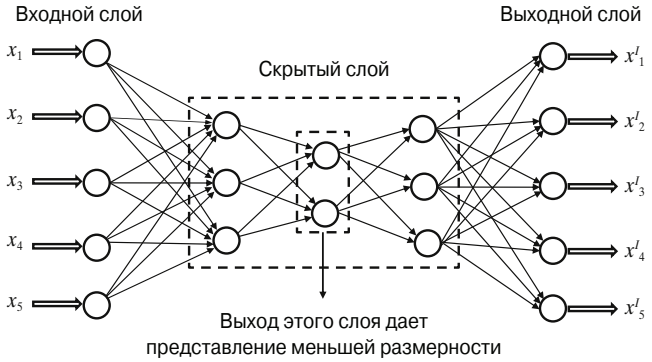


Рис. 1.12. Пример автокодировщика с несколькими выходами

Несмотря на то что полносвязная архитектура хорошо подходит для многих задач, ее эффективность часто удается повысить за счет исключения некоторых связей или их разделения тщательно продуманным способом. Как правило, понимание того, как это следует делать, приходит в результате анализа данных с учетом специфики конкретной задачи. Классическим примером такого исключения и разделения весов может служить *сверточная нейронная сеть* (глава 8), архитектура которой тщательно спроектирована таким образом, чтобы она согласовывалась с типичными свойствами данных изображения. Такой подход минимизирует риск *переобучения* путем учета специфики свойств данных этого типа (смещения). Как будет подробно обсуждаться в главе 4, переобучение — распространенная проблема нейронных сетей, проявляющаяся в том, что сеть очень хорошо работает на тренировочных данных, но плохо *обобщается* на неизвестные тестовые данные. Эта проблема возникает тогда, когда количество свободных параметров (обычно равное количеству взвешенных связей) слишком велико по сравнению с объемом тренировочных данных. В подобных случаях многочисленные параметры запоминают специфические нюансы тренировочных данных, но теряют способность распознавать статистически значимые закономерности и использовать их для классификации неизвестных тестовых данных. Совершенно очевидно, что увеличение количества узлов в нейронной сети повышает ее склонность к переобучению. В последнее время интенсивно ведется работа по совершенствованию как архитектур нейронных сетей, так и вычислений, выполняемых в каждом узле, направленная на то, чтобы минимизировать риски переобучения. К тому же на качество конечного решения оказывает влияние также способ, используемый для тренировки сети. За последние годы для повышения качества обучения был предложен ряд эффективных методов, таких как *предварительное обучение* (pretraining), о котором

будет рассказано в главе 4. Все эти современные методы тренировки подробно обсуждаются в данной книге.

1.2.3. Многослойная нейронная сеть как вычислительный граф

Нейронную сеть полезно рассматривать в качестве *вычислительного графа*, который строится путем объединения в единое целое многих базовых параметрических моделей. Нейронные сети обладают значительно более мощными возможностями, чем их строительные блоки, поскольку параметры этих моделей обучаются *совместному* созданию в высшей степени оптимизированной композиционной функции моделей. Обычное использование термина "перцептрон" в отношении базового элемента нейронной сети может вводить в заблуждение, поскольку разработаны многочисленные вариации этого базового элемента, являющиеся более предпочтительными в тех или иных ситуациях. В действительности в качестве строительных блоков таких моделей чаще встречаются логистические элементы (с сигмоидной функцией активации).

Многослойная сеть оценивает результат композиций функций, вычисляемых в индивидуальных узлах. Путь длиной 2 в нейронной сети, в которой вслед за функцией $f(\cdot)$ вычисляется функция $g(\cdot)$, можно рассматривать как сложную функцию $f(g(\cdot))$. Кроме того, если $g_1(\cdot)$, $g_2(\cdot)$, …, $g_k(\cdot)$ — это функции, вычисляемые в слое m, а отдельный узел слоя $(m + 1)$ вычисляет функцию $f(\cdot)$, то сложная функция, вычисляемая узлом слоя $(m + 1)$ в терминах входов слоя m, может быть записана как $f(g_1(\cdot)\ldots g_k(\cdot))$. Ключом к повышению вычислительной мощности множества слоев является использование нелинейной функции активации. Можно показать, что если во всех слоях используется *тождественная* (линейная) функция активации, то такая многослойная сеть сводится к линейной регрессии. Было доказано [208], что сеть, состоящая из одного скрытого слоя нелинейных элементов (с широким выбором *функций сжатия* наподобие сигмоиды) и одного (линейного) выходного слоя, способна вычислить почти любую "разумную" функцию. Вследствие этого нейронные сети часто называют *универсальными аппроксиматорами функций* (universal function approximators), хотя это теоретическое заявление не всегда легко реализовать на практике. Основная трудность состоит в том, что количество необходимых для этого скрытых элементов довольно велико, что увеличивает количество параметров, подлежащих обучению. В результате на практике при обучении сети с использованием ограниченного объема данных возникают проблемы. Фактически глубокие сети нередко оказываются более предпочтительными, поскольку они позволяют уменьшить количество скрытых элементов в каждом слое, а вместе с этим и общее количество параметров.

Использование выражения “строительный блок” применительно к многослойным нейронным сетям представляется особенно уместным. Очень часто готовые библиотеки[2], предназначенные для создания нейронных сетей, предоставляют аналитикам доступ к таким строительным блокам. Аналитику достаточно указать количество и тип элементов в каждом слое, а также готовую или нестандартную функцию потерь. Глубокую нейронную сеть, содержащую десятки слоев, часто можно описать несколькими строками кода. Обучение весов осуществляется автоматически с помощью алгоритма обратного распространения ошибки, который использует динамическое программирование для формирования сложных этапов обновления параметров базового вычислительного графа. Аналитик не должен тратить время и усилия на самостоятельное прохождение этих этапов, благодаря чему испытание различных типов архитектур будет восприниматься им как относительно безболезненный процесс. Создание нейронной сети с помощью готовых программных средств часто сравнивают со сборкой игрушек из деталей детского конструктора. Каждый блок подобен элементу (или слою элементов) с определенным типом активации. Значительное упрощение обучения нейронных сетей достигается за счет использования алгоритма обратного распространения ошибки, который избавляет аналитика от ручного проектирования всех этапов по обновлению параметров, являющегося на самом деле лишь частью чрезвычайно сложного процесса оптимизации. Прохождение этих этапов зачастую является наиболее сложной частью большинства алгоритмов машинного обучения, и важная роль парадигмы нейронных сетей заключается в привнесении идеи модульности в машинное обучение. Другими словами, модульность структуры нейронной сети транслируется в модульность процесса обучения ее параметров. Для последнего типа модульности существует специальное название — *обратное распространение ошибки* (backpropagation). При таком подходе характер проектирования нейронных сетей изменяется, и вместо занятия для математиков оно становится задачей для (опытных) инженеров.

1.3. Тренировка нейронной сети с помощью алгоритма обратного распространения ошибки

Процесс обучения однослойной нейронной сети отличается сравнительной простотой, поскольку ошибка (или функция потерь) может быть вычислена как непосредственная функция весов, что дает возможность легко вычислять градиент. В случае многослойных нейронных сетей проблема обусловлена тем, что потери представляют собой сложную композиционную функцию весов,

[2] В качестве примера можно привести библиотеки Torch [572], Theano [573] и TensorFlow [574].

относящихся к предыдущим слоям. Градиент композиционной функции вычисляется с помощью алгоритма обратного распространения ошибки. Этот алгоритм использует цепное правило дифференциального исчисления, в соответствии с которым градиенты вычисляются путем суммирования произведений локальных градиентов вдоль различных путей от узла к выходу. Несмотря на то что эта сумма включает экспоненциально возрастающее количество компонент (путей), она поддается эффективному вычислению с помощью *динамического программирования*. Алгоритм обратного распространения ошибки является непосредственным результатом динамического программирования и включает две фазы: *прямую* и *обратную*. Прямая фаза необходима для вычисления выходных значений и локальных производных в различных узлах, а обратная — для аккумуляции произведений этих локальных значений по всем путям, ведущим от данного узла к выходу.

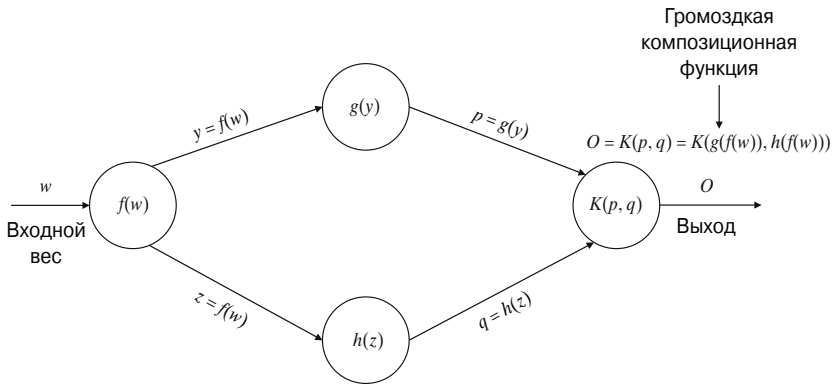
1. *Прямая фаза.* На протяжении этой фазы нейронной сети передаются входные данные тренировочного примера. Результатом является каскад вычислений, выполняемых поочередно в каждом из слоев в порядке их следования с использованием текущего набора весов. Предсказанный окончательный выход сравнивается с тренировочным примером, что позволяет вычислить для данного выхода производную функции потерь. Эта производная вычисляется по весам во всех слоях на протяжении обратной фазы.
2. *Обратная фаза.* Основной задачей обратной фазы является обучение градиенту функции потерь по различным весам посредством использования цепного правила дифференциального исчисления. Результирующие градиенты используются для обновления весов. Поскольку процесс тренировки (обучения) градиентов осуществляется в обратном направлении, его называют обратной фазой. Рассмотрим последовательность скрытых элементов $h_1, h_2, \ldots, h_k$, за которой следует выход o, относительно которого вычисляется функция потерь L. Кроме того, предположим, что вес соединения, связывающего скрытый элемент h_r с элементом h_{r+1} равен $w_{(h_r, h_{r+1})}$. Тогда в случае, если существует единственный путь, соединяющий узел h_1 с выходом o, градиент функции потерь по любому из этих весов может быть вычислен с помощью цепного правила в соответствии со следующей формулой:

$$\frac{\partial L}{\partial w_{(h_{r-1}, h_r)}} = \frac{\partial L}{\partial o} \cdot \left[\frac{\partial o}{\partial h_k} \prod_{i=r}^{k-1} \frac{\partial h_{i+1}}{\partial h_i} \right] \frac{\partial h_r}{\partial w_{(h_{r-1}, h_r)}} \quad \forall r \in 1 \ldots k. \tag{1.22}$$

В этом выражении предполагается, что в сети существует *только один путь*, ведущий от h_1 к o, тогда как в действительности число таких путей

может экспоненциально возрастать с ростом размера сети. Для расчета градиента в вычислительном графе, в котором существует не один такой путь, используют обобщенный вариант цепного правила, так называемое *многомерное цепное правило* (multivariable chain rule). Это достигается добавлением композиций вдоль каждого из путей, ведущих от узла h_1 к o. Пример, иллюстрирующий применение цепного правила в вычислительном графе с двумя путями, приведен на рис. 1.13. Поэтому представленное выше выражение обобщают на случай, когда существует набор P путей, ведущих от узла h_r к o:

$$\frac{\partial L}{\partial w_{(h_{r-1},\, h_r)}} = \underbrace{\frac{\partial L}{\partial o} \cdot \left[\sum_{[h_r,\, h_{r+1},\, \ldots,\, h_{k,\, o}] \in P} \frac{\partial o}{\partial h_k} \prod_{i=r}^{k-1} \frac{\partial h_{i+1}}{\partial h_i} \right]}_{\substack{\text{Алгоритм обратного распространения} \\ \text{ошибки вычисляет } \nabla(h_r, o) = \dfrac{\partial L}{\partial h_r}}} \frac{\partial h_r}{\partial w_{(h_{r-1},\, h_r)}}. \tag{1.23}$$



$$\frac{\partial o}{\partial w} = \frac{\partial o}{\partial p} \cdot \frac{\partial p}{\partial w} + \frac{\partial o}{\partial q} \cdot \frac{\partial q}{\partial w} = \qquad \text{[многомерное цепное правило]}$$

$$= \frac{\partial o}{\partial p} \cdot \frac{\partial p}{\partial y} \cdot \frac{\partial y}{\partial w} + \frac{\partial o}{\partial q} \cdot \frac{\partial q}{\partial z} \cdot \frac{\partial z}{\partial w} = \qquad \text{[одномерное цепное правило]}$$

$$= \underbrace{\frac{\partial K(p,\, q)}{\partial p} \cdot g'(y) \cdot f'(w)}_{\text{Первый путь}} + \underbrace{\frac{\partial K(p,\, q)}{\partial q} \cdot h'(z) \cdot f'(w).}_{\text{Второй путь}}$$

Рис. 1.13. Цепное правило в вычислительных графах. Произведения специфических для узлов частных производных вдоль путей от веса w к выходу o агрегируются. Результирующее значение дает производную выхода o по весу w. В данном простом примере существуют лишь два пути, ведущих от выхода к входу

Вычисление члена $\frac{\partial h_r}{\partial w_{(h_{r-1},\, h_r)}}$ в правой части, к которому мы еще вернемся (уравнение 1.27), не представляет трудностей. Однако член $\nabla(h_r,\, o) = \frac{\partial L}{\partial h_r}$ агрегируется по экспоненциально возрастающему (с ростом длины пути) количеству путей, что, на первый взгляд, кажется непреодолимым препятствием. В данном случае ключевую роль играет тот факт, что вычислительный граф нейронной сети не имеет циклов, что делает принципиально возможным вычисление подобной агрегации в обратном направлении путем вычисления сначала членов D(h_k, o) для узлов h_k, ближайших к o, а затем рекурсивного вычисления этих значений для узлов в предыдущих слоях по отношению к последующим. Кроме того, значение D(o, o) для каждого выходного узла инициализируется следующим образом:

$$\Delta(o,\, o) = \frac{\partial L}{\partial o}. \tag{1.24}$$

Этот тип динамического программирования часто используют для эффективного вычисления всех типов функций, центрированных на путях в направленных ациклических графах, что в противном случае потребовало бы экспоненциально возрастающего количества операций. Рекурсию для D(h_k, o) можно легко получить с помощью многомерного цепного правила:

$$\Delta(h_r,\, o) = \frac{\partial L}{\partial h_r} = \sum_{h:h_r \Rightarrow h} \frac{\partial L}{\partial h}\frac{\partial h}{\partial h_r} = \sum_{h:h_r \Rightarrow h} \frac{\partial h}{\partial h_r}\Delta(h,\, o). \tag{1.25}$$

Поскольку все фигурирующие здесь узлы h содержатся в слоях, следующих за слоем, в котором находится узел h_r, то к моменту вычисления члена D(h_k, o) член D(h, o) оказывается уже вычисленным. Однако для вычисления уравнения 1.25 нам все еще нужно вычислить член $\frac{\partial h}{\partial h_r}$. Рассмотрим ситуацию, в которой связь, соединяющая узел h_r с узлом h, имеет вес $w_{(h_r,\, h)}$, и пусть a_h — значение, вычисленное в скрытом элементе h непосредственно *перед* применением функции активации Φ(·). Иными словами, мы имеем соотношение $h = \Phi(a_h)$, где a_h — линейная комбинация входов элемента h, связанных с элементами предыдущего слоя. Тогда в соответствии с одномерным цепным правилом мы получаем для члена следующее выражение:

$$\frac{\partial h}{\partial h_r} = \frac{\partial h}{\partial a_h}\cdot\frac{\partial a_h}{\partial h_r} = \frac{\partial \Phi(a_h)}{\partial a_h}\cdot w_{(h_r,\, h)} = \Phi'(a_h)\cdot w_{(h_r,\, h)}.$$

Это значение $\frac{\partial h}{\partial h_r}$ используется в уравнении 1.25, которое рекурсивно повторяется в обратном направлении, начиная с выходного узла. Соответствующие обновления в обратном направлении можно записать следующим образом:

$$\Delta(h_r,\ o) = \sum_{h:h_r \Rightarrow h} \Phi'(a_h) \cdot w_{(h_r,\ h)} \cdot \Delta(h,\ o). \tag{1.26}$$

Поэтому градиенты последовательно аккумулируются при проходе в обратном направлении, и при этом каждый узел обрабатывается ровно один раз. Обратите внимание на то, что для вычисления градиента по всем весам связей уравнение 1.25 (число операций в котором пропорционально числу исходящих связей) должно быть вычислено для каждой входящей связи узла. Наконец, уравнение 1.23 требует вычисления члена $\frac{\partial h_r}{\partial w_{(h_{r-1},\ h_r)}}$, что можно легко сделать в соответствии со следующей формулой:

$$\frac{\partial h_r}{\partial w_{(h_{r-1},\ h_r)}} = h_{r-1} \cdot \Phi'(a_{h_r}). \tag{1.27}$$

Здесь основным градиентом, распространяющимся в обратном направлении, является производная по активациям слоя, а градиент по весам легко вычисляется для любой входящей связи соответствующего элемента.

Следует отметить, что динамическая рекурсия (уравнение 1.26) может вычисляться множеством способов, в зависимости от того, какие переменные используются в качестве промежуточных при формировании цепочек. Все подобные рекурсии эквиваленты в плане получения окончательного результата обратного распространения ошибки. Далее мы представим альтернативную версию такой рекурсии, которая чаще встречается в учебниках. Обратите внимание на то, что в уравнении 1.23 переменные в скрытых слоях используются в качестве "цепочечных" переменных для динамической рекурсии. Для цепного правила также можно использовать предактивационные переменные, которые в нейроне получают после применения линейного преобразования (но до применения активации) в качестве промежуточных переменных. Предактивационное значение скрытой переменной $h = \Phi(a_h)$ равно a_h. Различия между предактивационным и постактивационным значениями в нейроне показаны на рис. 1.7. Поэтому вместо уравнения 1.23 можно использовать следующее цепное правило:

$$\frac{\partial L}{\partial w_{(h_{r-1},\ h_r)}} = \underbrace{\frac{\partial L}{\partial o} \cdot \Phi'(a_o) \cdot \left[\sum_{[h_r,\ h_{r+1},\ \ldots,\ h_k,\ o] \in P} \frac{\partial a_o}{\partial a_{h_k}} \prod_{i=r}^{k-1} \frac{\partial a_{h_{i+1}}}{\partial a_{h_i}} \right]}_{\substack{\text{Алгоритм обратного распространения} \\ \text{ошибки вычисляет } \delta(h_r,\ o) = \frac{\partial L}{\partial a_{h_r}}}} \underbrace{\frac{\partial a_{h_r}}{\partial w_{(h_{r-1},\ h_r)}}}_{h_{r-1}}. \tag{1.28}$$

Записывая это рекурсивное уравнение, мы использовали обозначение $\delta(h_r,\ o) = \dfrac{\partial L}{\partial a_{h_r}}$ вместо $\Delta(h_r,\ o) = \dfrac{\partial L}{\partial h_r}$. Значение $\delta(o,\ o) = \dfrac{\partial L}{\partial a_o}$ инициализируется следующим образом:

$$\delta(o,\ o) = \frac{\partial L}{\partial a_o} = \Phi'(a_o) \cdot \frac{\partial L}{\partial o}. \tag{1.29}$$

Затем можно использовать многомерное цепное правило для записи аналогичной рекурсии:

$$\delta(h_r,\ o) = \frac{\partial L}{\partial a_{h_r}} = \sum_{h:h_r \Rightarrow h} \overbrace{\frac{\partial L}{\partial a_h}}^{\delta(h,\ o)} \underbrace{\frac{\partial a_h}{\partial a_{h_r}}}_{\Phi'(a_{h_r})w_{(h_r,\ h)}} = \Phi'(a_{h_r}) \sum_{h:h_r \Rightarrow h} w_{(h_r,\ h)} \cdot \delta(h,\ o). \tag{1.30}$$

Именно в такой форме условие рекурсии чаще всего приводится в учебниках при обсуждении механизма обратного распространения ошибки. Зная $d(h_r, o)$, мы можем вычислить частную производную функции потерь по весу с помощью следующей формулы:

$$\frac{\partial L}{\partial w_{(h_{r-1},\ h_r)}} = \delta(h_r,\ o) \cdot h_{r-1}. \tag{1.31}$$

Как и в случае однослойной сети, процесс обновления узлов осуществляется эпохами, на протяжении каждой из которых выполняется обработка всех тренировочных данных, пока не будет достигнута сходимость. Алгоритм обратного распространения ошибки подробно рассматривается в главе 3. В данной главе мы ограничимся лишь кратким обсуждением этой темы.

1.4. Практические аспекты тренировки нейронных сетей

Несмотря на то что нейронные сети пользуются репутацией универсальных аппроксиматоров функций, по-прежнему существуют значительные трудности, связанные с их фактической тренировкой, которая обеспечивала бы такой уровень их функциональности. Эти трудности в основном обусловлены некоторыми практическими проблемами процесса тренировки, наиболее важной из которых является проблема *переобучения* (overfitting).

1.4.1. Проблема переобучения

Суть проблемы переобучения заключается в том, что подгонка модели под конкретный тренировочный набор данных еще не гарантирует, что она будет хорошо предсказывать тестовые данные, которые ей прежде не встречались, даже если на тренировочных данных она идеально с этим справлялась. Иными словами, между качеством работы сети с тренировочными и тестовыми данными всегда существует брешь, размеры которой увеличиваются в случае сложных моделей и небольших наборов данных.

Чтобы понять природу описанного явления, рассмотрим простую однослойную сеть, для обучения которой вещественным целевым значениям из набора данных, имеющих пять атрибутов, мы будем использовать тождественную (линейную) функцию активации. Эта архитектура почти идентична той, которая представлена на рис. 1.3, если отвлечься от того, что в данном случае предсказываются вещественные значения. Поэтому сеть пытается обучить следующую функцию:

$$\hat{y} = \sum_{i=1}^{5} w_i \cdot x_i. \tag{1.32}$$

Рассмотрим ситуацию, в которой наблюдаемое целевое значение является вещественным числом и всегда в два раза превышает значение первого атрибута, тогда как значения остальных атрибутов никак не связаны с целевым значением. В то же время мы имеем в своем распоряжении лишь четыре тренировочных примера, т.е. их количество на единицу меньше количества признаков (свободных параметров). Для конкретности предположим, что мы работаем со следующим набором тренировочных примеров:

x_1	x_2	x_3	x_4	x_5	y
1	1	0	0	0	2
2	0	1	0	0	4
3	0	0	1	0	6
4	0	0	0	1	8

В данном случае, учитывая известное соотношение между первым признаком и целевым значением, корректным вектором параметров будет вектор $\overline{W} = [2,\ 0,\ 0,\ 0,\ 0]$. Тренировочный набор обеспечивает нулевую ошибку для данного решения, хотя сеть должна *обучиться* ему на примерах, поскольку оно не задано нам априори. Однако при этом мы сталкиваемся с той проблемой, что количество тренировочных точек меньше количества параметров, а это означает, что в данном случае можно найти бесчисленное количество решений с нулевой ошибкой. Например, набор параметров [0, 2, 4, 6, 8] также обеспечивает

нулевую ошибку на *тренировочных данных*. Но если мы применим это решение к неизвестным тестовым данным, то сеть, вероятнее всего, будет работать очень плохо, поскольку мы просто удачно подобрали параметры для одного случая и они не будут хорошо *обобщаться* на новые точки, т.е. обеспечивать целевое значение, в два раза превышающее значение первого атрибута, при произвольных значениях остальных атрибутов. К этому типу ложного вывода приводит малочисленность тренировочных данных, когда в модели кодируются случайные нюансы. В результате решение плохо обобщается на неизвестные тестовые данные. Эта ситуация почти аналогична обучению путем простого запоминания, которое характеризуется высокой предсказательной силой для тренировочных данных, но неспособно предсказывать правильный результат для неизвестных данных. Увеличение количества тренировочных примеров улучшает обобщающую способность модели, тогда как повышение сложности модели снижает ее способность к обобщению. В то же время, если набор тренировочных данных достаточно велик, то маловероятно, что слишком простой модели удастся уловить сложные соотношения, существующие между признаками и целевыми значениями. Простое практическое правило заключается в том, что количество тренировочных точек данных должно превышать количество параметров по крайней мере в 2-3 раза, хотя точное количество необходимых примеров определяется спецификой конкретной задачи. В общем случае о моделях с большим количеством параметров говорят как о моделях *большой мощности*, и для того, чтобы такая модель хорошо обобщалась на неизвестные тестовые данные, объемы данных, используемых для ее тренировки, также должны быть большими. Понятие переобучения тесно связано с достижением компромисса между смещением и дисперсией алгоритма машинного обучения. *Смещение* (bias) алгоритма является показателем того, насколько хорошо он аппроксимирует зависимость между данными и ответами, тогда как *дисперсия* (variance) алгоритма — это показатель степени разброса ответов в зависимости от изменения выборки данных. Суть вывода, который следует из анализа баланса этих показателей, заключается в том, что использование более мощных (т.е. менее *смещенных*) моделей не всегда обеспечивает выигрыш при работе с ограниченными объемами тренировочных данных в силу более высокой дисперсии таких моделей. Например, если вместо тренировочных данных, приведенных в таблице выше, мы используем другой набор, состоящий из четырех точек данный, то, скорее всего, получим в результате обучения сети совершенно другой набор параметров (основанный на случайных различиях между этими точками). Применение новой модели к *тому же тестовому примеру* приведет, скорее всего, к предсказаниям, которые будут отличаться от прежних, полученных при использовании первого тренировочного набора. Этот тип разброса предсказаний для одного и того же тестового примера и есть проявление *дисперсии модели*,

которая вносит свой вклад в ошибку. В конце концов, два разных предсказания, полученных для одного и того же тестового набора, не могут быть корректными одновременно. Недостатком сложных моделей является то, что они видят ложные закономерности в случайных нюансах, что становится особенно заметным при недостаточности тренировочных данных. Принимая решения относительно сложности модели, необходимо держаться золотой середины. Все эти вопросы подробно рассматриваются в главе 4.

Нейронные сети всегда считались инструментом, способным, по крайней мере теоретически, аппроксимировать любую функцию [208]. Но в случае недостаточности данных сеть может работать плохо; это и есть одна из причин того, почему нейронные сети приобрели широкую известность лишь в последнее время. Облегчение доступа к данным обнаружило преимущества нейронных сетей по сравнению с традиционным машинным обучением (см. рис. 1.2). В целом же нейронные сети должны тщательно проектироваться для минимизации отрицательных эффектов переобучения даже при наличии доступа к большим объемам данных. В этом разделе дается обзор некоторых методов проектирования, используемых для уменьшения влияния переобучения.

1.4.1.1. Регуляризация

Поскольку увеличение количества параметров повышает вероятность переобучения, вполне естественно ограничиваться использованием меньшего количества ненулевых параметров. Если в предыдущем примере ограничиться вектором $\bar{W}$, имеющим лишь одну ненулевую компоненту из пяти возможных, то это обеспечит получение корректного решения [2, 0, 0, 0, 0]. Уменьшение абсолютных значений параметров также способствует ослаблению эффектов переобучения. Ввиду трудностей ограничения значений параметров прибегают к более мягкому подходу, заключающемуся в добавлении штрафного члена $\lambda \left\| \bar{W} \right\|^p$ в используемую функцию потерь. Обычно p задают равным 2, что соответствует *регуляризации Тихонова*. В общем случае в целевую функцию добавляется квадрат значения каждого параметра (умноженный на параметр регуляризации $\lambda > 0$). На практике это изменение приводит к тому, что из обновления параметра w_i вычитается величина, пропорциональная lw_i. В качестве примера ниже приведена регуляризированная версия уравнения 1.6 для мини-пакета S и шага обновления $\alpha > 0$:

$$\bar{W} \Leftarrow \bar{W}(1 - \alpha\lambda) + \alpha \sum_{\bar{X} \in S} E(\bar{X})\bar{X}. \tag{1.33}$$

Здесь $E(\bar{X})$ представляет текущую ошибку $(y - \hat{y})$, т.е. разницу между наблюдаемым и предсказанным значениями для тренировочного примера $\bar{X}$. Такой тип штрафования можно рассматривать как своего рода *ослабление веса* в

процессе обновления. Регуляризация особенно важна в условиях ограниченности доступного объема данных. Хорошей аналогией регуляризации из области биологии может служить процесс постепенного забывания, заключающийся в том, что "менее важные" (т.е. *шум*) связи между данными не сохраняются. В общем случае часто целесообразно использовать более сложные модели с регуляризацией, а не простые модели без регуляризации.

Попутно заметим, что общая форма уравнения 1.33 используется многими регуляризированными моделями машинного обучения, такими как *регрессия по методу наименьших квадратов* (глава 2), где вместо $E(\overline{X})$ подставляется функция ошибки, специфическая для данной модели. Интересен тот факт, что в однослойном перцептроне ослабление весов используют лишь в редких случаях[3], поскольку иногда это может чрезмерно ускорять процесс забывания, если в векторе весов доминирует небольшое количество неверно классифицированных тренировочных точек. Основная проблема состоит в том, что критерий перцептрона (в отличие от линейно-кусочной или квадратичной функций потерь) уже сам по себе является вырожденной функцией потерь с минимальным значением 0 при $\overline{W} = 0$. Эта особенность является следствием того факта, что однослойный перцептрон первоначально был определен в терминах обновлений, имитирующих биологические нейроны, а не в терминах тщательно продуманной функции потерь. За исключением линейно разделимых классов, о гарантиях сходимости к оптимальному решению речь никогда не шла. Обычно для однослойного перцептрона используются другие методы регуляризации, которые обсуждаются ниже.

1.4.1.2. Архитектура нейрона и разделение параметров

Наиболее эффективный подход к построению нейронной сети заключается в том, чтобы приступать к разработке ее архитектуры лишь после тщательного изучения особенностей базовых данных. Знание этих особенностей позволяет создавать специализированные архитектуры с меньшим количеством параметров. Помимо этого, многие параметры могут разделяться. Например, сверточная нейронная сеть использует один и тот же набор параметров для обучения характеристикам локального блока изображения. Последние достижения в разработке *рекуррентных* и *сверточных нейронных сетей* наглядно демонстрируют плодотворность такого подхода.

1.4.1.3. Раннее прекращение обучения

Другой распространенной формой регуляризации является *раннее прекращение обучения*, когда процесс градиентного спуска останавливают после

[3] Обычно в однослойных моделях, а также в многослойных моделях с большим количеством параметров ослабление весов используется с другими функциями потерь.

нескольких итераций. Один из способов определения точки останова заключается в том, чтобы удержать часть тренировочных данных, а затем тестировать ошибку модели на этом удержанном наборе. Процесс градиентного спуска прекращается, когда ошибка на удержанном наборе начинает возрастать. Раннее прекращение обучения снижает размер пространства параметров до меньшей окрестности в пределах начальных значений параметров. С этой точки зрения раннее прекращение обучения действует в качестве регуляризатора, поскольку оно в конечном счете ограничивает пространство параметров.

1.4.1.4. Достижение компромисса между шириной и глубиной

Как ранее уже обсуждалось, при наличии в скрытом слое большого количества скрытых элементов двухслойная нейронная сеть может использоваться в качестве универсального аппроксиматора функций [208]. Оказывается, что в случае нейронных сетей с еще б*о*льшим количеством слоев (т.е. обладающих *большей глубиной*) обычно требуется меньшее количество элементов, приходящихся на один слой, поскольку композиционные функции, создаваемые последовательными слоями, повышают мощность нейронной сети. Увеличение глубины сети — это разновидность регуляризации, поскольку признаки в последующих слоях вынуждены подчиняться конкретному типу структуры, навязываемому предшествующими слоями. Усиление ограничений уменьшает мощность сети, что приносит пользу, если существуют ограничения на объем доступных данных. Краткое объяснение такого поведения приведено в разделе 1.5. Обычно количество элементов в каждом слое может быть уменьшено до такого уровня, что нередко глубокая сеть будет иметь гораздо меньшее количество параметров даже при увеличении количества слоев. Это наблюдение стало причиной бума исследований в области *глубокого обучения* (deep learning).

Несмотря на то что глубокие сети позволили снизить остроту проблемы переобучения, они привнесли проблемы другого рода, осложняющие обучение сети. В частности, производные функции потерь по весам в разных слоях имеют, как правило, значительно различающиеся величины, что затрудняет выбор наиболее оптимальных размеров шагов. Это нежелательное поведение порождает проблемы так называемых *затухающих* и *взрывных* градиентов. Кроме того, для достижения сходимости в глубоких сетях часто требуется недопустимо длительное время. Эти проблемы еще будут обсуждаться в этой и последующих главах.

1.4.1.5. Ансамблевые методы

Для повышения обобщающей способности модели используют различные ансамблевые методы, такие как метод *параллельной композиции алгоритмов*,

или *бэггинг* (англ. “bagging” от “*b*ootstrap *agg*regat*ing*” — улучшенное агрегирование). Эти методы применимы не только к нейронным сетям, но и к любому типу алгоритмов машинного обучения. Однако в последние годы был предложен ряд ансамблевых методов, предназначенных для нейронных сетей. К их числу относятся метод исключения, или *дропаут* (dropout), и метод *прореживания связей* (dropconnect). Эти методы могут сочетаться со многими архитектурами нейронных сетей, обеспечивая дополнительное повышение точности около 2% во многих реальных конфигурациях. Однако повышение точности зависит от типа данных и природы базовой тренировки. Например, нормализация активации в скрытых слоях может снизить эффективность методов исключения, но при этом сама активация может обеспечить результирующий выигрыш. Ансамблевые методы обсуждаются в главе 4.

1.4.2. Проблемы затухающих и взрывных градиентов

В то время как увеличение глубины сети во многих случаях позволяет уменьшить количество параметров, это приводит к появлению ряда затруднений практического характера. Слабой стороной использования метода обратного распространения ошибки с помощью цепного правила в сетях с большим количеством слоев является нестабильность обновлений. В частности, в некоторых типах архитектур обновления в начальных слоях нейронной сети могут иметь либо пренебрежимо малую (затухающий градиент), либо непомерно большую (взрывной градиент) величину. В основном это обусловлено поведением члена в виде произведения в уравнении 1.23, который может либо экспоненциально возрастать, либо экспоненциально убывать с увеличением длины пути. Чтобы понять природу этого явления, рассмотрим ситуацию, когда имеется многослойная сеть с одним нейроном в каждом слое. Можно показать, что каждая локальная производная вдоль пути представляет собой произведение веса и производной функции активации. Общая производная для обратного распространения равна произведению этих значений. Если каждое значение подчиняется случайному распределению с математическим ожиданием меньше 1, то произведение этих производных в уравнении 1.23 будет экспоненциально затухать с увеличением длины пути. Если же математические ожидания индивидуальных значений вдоль пути больше 1, то в типичных случаях это приводит к “взрывному” поведению градиента, т.е. его безудержному росту. Даже если значения локальных производных подчинены случайному распределению с математическим ожиданием, в точности равным единице, то общая производная, как правило, будет демонстрировать нестабильное поведение, в зависимости от фактического распределения указанных значений. Другими словами, возникновение проблем затухающих и взрывных градиентов — довольно естественное явление для глубоких сетей, что делает процесс их тренировки нестабильным.

Для преодоления этих проблем было предложено не одно решение. Так, проблема затухающих градиентов характерна для сигмоиды, поскольку ее производная меньше 0,25 при всех значениях аргумента (упражнение 7) и имеет предельно малые значение в области насыщения. Элемент активации ReLU менее подвержен проблеме затухающего градиента, поскольку его производная всегда равна 1 для положительных значений аргумента. Более подробно эти вопросы обсуждаются в главе 3. Помимо использования ReLU, для улучшения сходимости градиентного спуска существует множество других способов. В частности, во многих случаях для этого применяют *адаптивные скорости обучения* и *методы сопряженных градиентов*. Кроме того, для преодоления некоторых из описанных выше проблем можно воспользоваться *пакетной нормализацией* (batch normalization). Обо всем этом речь пойдет в главе 3.

1.4.3. Трудности со сходимостью

Добиться достаточно быстрой сходимости процесса оптимизации в случае очень глубоких сетей довольно трудно, поскольку увеличение глубины сети повышает ее сопротивляемость процессу тренировки, нарушая плавное перетекание градиента по сети. Эта проблема в некотором смысле родственна проблеме затухающего градиента, но имеет собственные уникальные характеристики. Специально для таких случаев в литературе был предложен ряд “трюков”, включая использование *управляемых (вентильных) сетей* (gating networks) и *остаточных сетей* (residual networks) [184]. Эти методы обсуждаются в главах 7 и 8 соответственно.

1.4.4. Локальные и ложные оптимумы

Оптимизируемая функция нейронной сети является в высшей степени нелинейной и имеет множество локальных оптимумов. В случае большого пространства параметров с многочисленными локальными оптимумами имеет смысл потратить некоторое время на выбор начальных точек. Одним из способов улучшения инициализации параметров нейронной сети является *предварительное обучение* (pretraining). Основная идея заключается в обучении (с учителем или без учителя) *мелких подсетей* исходной сети для создания начальных значений весов. Этот тип предварительного обучения применяется *послойно в жадной манере*, т.е. все слои тренируются поочередно для обучения начальным значениям своих параметров. Такой подход позволяет избежать попадания начальных точек в явно неподходящие области пространства параметров. Кроме того, предварительное обучение без учителя часто способствует устранению проблем переобучения. Основная идея заключается в том, что некоторые минимумы функции потерь являются ложными оптимумами, поскольку они проявляются лишь на тренировочных данных и не проявляются на тестовых. Как

правило, обучение без учителя смещает начальную точку в сторону бассейнов аттракции "хороших" оптимумов для тестовых данных. Этот вопрос также связан с обобщаемостью модели. Методы предварительного обучения обсуждаются в разделе 4.7.

Интересно отметить, что понятие ложного минимума часто рассматривают сквозь призму способности модели нейронной сети к обобщению. Ход связанных с этим рассуждений отличается от того, который принят в традиционных методах оптимизации. Традиционные методы оптимизации фокусируются не на различиях в функциях потерь тренировочных и тестовых данных, а только на форме функции потерь тренировочных данных. Как это ни удивительно, проблема локальных оптимумов (с точки зрения традиционных подходов) в нейронных сетях в действительности не столь серьезна, как мы могли бы ожидать от таких нелинейных функций. В большинстве случаев нелинейность порождает проблемы, связанные с собственно процессом тренировки (например, отсутствие сходимости), а не с "застреванием" в локальном минимуме.

1.4.5. Вычислительные трудности

Важным фактором, который следует учитывать при проектировании нейронной сети, является время, необходимое для ее тренировки. В области распознавания текстов и изображений сети нередко тренируют неделями. Достигнутый в последние годы прогресс в разработке такого оборудования, как графические процессоры (GPU), способствовал значительному прорыву в данном направлении. GPU — это специализированные процессоры, позволяющие резко ускорить выполнение операций, типичных для нейронных сетей. В этом смысле особенно удобными оказываются такие алгоритмические фреймворки, как *Torch*, поскольку они предоставляют поддержку GPU, тесно интегрированную в платформу.

Несмотря на то что прогресс в разработке алгоритмов сыграл свою роль в возбуждении интереса к глубокому обучению, даже прежние алгоритмы способны дать нам гораздо больше, если они выполняются на современном оборудовании. В свою очередь, более скоростное оборудование стимулирует разработку алгоритмов, поскольку этот процесс изначально требует многократного выполнения интенсивных в вычислительном отношении тестов. Например, последний вариант такой модели, как *долгая краткосрочная память* (long short-term memory), претерпел лишь умеренные изменения [150] с тех пор, как в 1997 году был предложен ее первоначальный вариант [204]. И все же потенциальные возможности этой модели были распознаны лишь недавно благодаря повышению вычислительной мощности современных компьютеров и появлению алгоритмических новинок, разработка которых стала возможной в силу создания лучших условий для проведения экспериментов.

Удобным свойством преобладающего большинства моделей нейронных сетей является то, что бо́льшая часть тяжелой вычислительной работы переносится в фазу тренировки, тогда как для фазы предсказания нередко характерна более высокая эффективность в вычислительном отношении, поскольку она требует выполнения меньшего количества операций (в зависимости от количества слоев). Это очень важный момент, поскольку фактор времени нередко более критичен для фазы предсказания, чем для фазы тренировки. Например, очень важно иметь возможность классифицировать изображение в режиме реального времени (с помощью предварительно созданной модели), даже если для фактического создания этой модели потребуется несколько недель обрабатывать миллионы изображений. Также были разработаны методы сжатия обученных сетей, обеспечивающие возможность их развертывания на мобильных и миниатюрных устройствах. Эти вопросы обсуждаются в главе 3.

1.5. Преимущества композиции функций

Несмотря на всю соблазнительность попыток привлечения биологических метафор для того, чтобы попытаться понять на интуитивном уровне, в чем кроется секрет удивительных вычислительных возможностей нейронных сетей, они не позволяют нам получить полную картину ситуаций, в которых нейронные сети хорошо работают. На самом низком уровне абстракции нейронную сеть можно рассматривать как вычислительный граф, комбинирующий простые функции для получения более сложной функции. Мощь глубокого обучения во многом объясняется тем фактом, что *повторная* композиция многочисленных нелинейных функций обладает значительной выразительной силой. Несмотря на то что в работе [208] было показано, что однократная композиция большого количества *функций сжатия* позволяет аппроксимировать почти любую функцию, такой подход требует использования чрезвычайно большого количества элементов (а значит, и параметров) сети. Это увеличивает емкость сети, что приводит к переобучению, избавиться от которого можно только предельным увеличением размера набора данных. Мощь глубокого обучения в значительной степени проистекает от того, что *многократная композиция определенных типов функций повышает репрезентативную способность сети, тем самым уменьшая размеры пространства параметров, подлежащих обучению*.

Не все базовые функции в одинаковой степени пригодны для достижения этой цели. В действительности нелинейные функции сжатия, используемые в нейронных сетях, не выбираются произвольно, а тщательно конструируются с учетом определенных типов свойств. Представьте, например, ситуацию, когда во всех слоях используется тождественная функция активации, так что вычисляются только линейные функции. В этом случае результирующая нейронная сеть будет не мощнее однослойной линейной сети.

Теорема 1.5.1. *Многослойная сеть, во всех слоях которой используется только тождественная функция активации, сводится к однослойной сети, выполняющей линейную регрессию.*

Доказательство. Рассмотрим сеть, которая содержит k скрытых слоев, а значит, в общей сложности $(k + 1)$ вычислительных слоев (включая выходной слой). Соответствующие $(k + 1)$ матриц весов связей между последовательными слоями обозначим как $W_1 \ldots W_{k+1}$. Пусть $\overline{x}$ — d-мерный вектор-столбец, соответствующий входу, $\overline{h}_1 \ldots \overline{h}_k$ — вектор-столбцы, соответствующие скрытым слоям, а $\overline{o}$ — m-мерный вектор-столбец, соответствующий выходу. Тогда для многослойных сетей справедливо следующее условие рекурсии:

$$\overline{h}_1 = \Phi(W_1^T \overline{x}) = W_1^T \overline{x},$$
$$\overline{h}_{p+1} = \Phi(W_{p+1}^T \overline{h}_p) = W_{p+1}^T \overline{h}_p \ \forall p \in \{1 \ldots k-1\},$$
$$\overline{o} = \Phi(W_{k+1}^T \overline{h}_k) = W_{k+1}^T \overline{h}_k.$$

Во всех вышеприведенных случаях в качестве функции активации $\Phi(\cdot)$ устанавливается тождественная функция. Путем исключения переменных скрытых слоев нетрудно получить следующее соотношение:

$$\overline{o} = W_{k+1}^T W_k^T \ldots W_1^T \overline{x} =$$
$$= \underbrace{(W_1 W_2 \ldots W_{k+1})^T}_{W_{xo}^T} \overline{x}.$$

Заметьте, что матрицу $W_1 W_2 \ldots W_{k+1}$ можно заменить новой матрицей W_{xo} размера $d \times m$ и обучать коэффициенты матрицы W_{xo}, а не коэффициенты всех матриц $W_1 W_2 \ldots W_{k+1}$, не теряя выразительности. Другими словами, получаем следующее соотношение:

$$\overline{o} = W_{xo}^T \overline{x}.$$

Однако это условие в точности идентично условию линейной регрессии с множественными выходами [6]. В действительности обучать сеть многочисленным матрицам $W_1 W_2 \ldots W_{k+1}$, а не матрице W_{xo}, — плохая идея, поскольку это увеличивает количество обучаемых параметров, но при этом никак не увеличивает мощность сети. Таким образом, многослойная нейронная сеть с тождественными функциями активации не дает никакого выигрыша в смысле выразительности по сравнению с однослойной сетью. Мы сформулировали этот результат для модели линейной регрессии с числовыми целевыми переменными. Аналогичный результат также справедлив для бинарных целевых переменных. В том специальном случае, когда во всех слоях используется тождественная функция активации, а в последнем слое для предсказаний используется единственный

выход со знаковой функцией активации, многослойная нейронная сеть сводится к перцептрону.

Лемма 1.5.1. *Рассмотрим многослойную сеть, во всех скрытых слоях которой используется тождественная активация, а единственный выходной узел использует критерий перцептрона в качестве функции потерь и знаковую функцию для получения предсказаний. Тогда такая нейронная сеть сводится к однослойному перцептрону.*

Данное утверждение доказывается почти так же, как и предыдущее. Фактически, при условии что скрытые слои являются линейными, введение дополнительных слоев не дает никакого выигрыша.

Этот результат показывает, что в целом использовать глубокие сети имеет смысл только тогда, когда функции активации в промежуточных слоях являются нелинейными. В качестве типичных примеров *функций сжатия* (squashing functions), которые "втискивают" выход в ограниченный интервал значений и градиенты которых максимальны вблизи нулевых значений, можно привести сигмоиду и гиперболический тангенс. При больших абсолютных значениях своих аргументов эти функции, как говорят, достигают *насыщения* в том смысле, что дальнейшее увеличение абсолютной величины аргумента не приводит к сколь-нибудь существенному изменению значения функции. Этот тип функций, значения которых не изменяются существенно при больших абсолютных значениях аргументов, разделяется другим семейством функций, так называемыми *гауссовскими ядрами* (Gaussian kernels), которые обычно используются в непараметрических оценках плотности распределения:

$$\Phi(v) = exp(-v^2 / 2). \tag{1.34}$$

Единственным отличием гауссовских ядер является то, что они насыщаются до нуля при больших значениях своих аргументов, в то время как сигмоида и гиперболический тангенс могут насыщаться до значений +1 и −1. В литературе, посвященной оценкам плотности, хорошо известен тот факт, что сумма многих небольших гауссовских ядер может быть использована для аппроксимации любой функции плотности [451]. Функции плотности имеют специальную неотрицательную структуру, в которой экстремумы распределения данных всегда насыщаются до нулевой плотности, и поэтому то же поведение демонстрируют и базовые ядра. В отношении функций сжатия действует (в более общем смысле) аналогичный принцип, в соответствии с которым любую функцию можно аппроксимировать некоторой линейной комбинацией небольших функций активации. В то же время функции сжатия не насыщаются до нуля, что дает возможность обрабатывать произвольное поведение при экстремальных значениях. Универсальная формулировка возможностей аппроксимации функций

нейронными сетями [208] гласит, что использование линейных комбинаций сигмоидных элементов (и/или большинства других разумных функций сжатия) в одном скрытом слое обеспечивает удовлетворительную аппроксимацию любой функции. Обратите внимание на то, что такая линейная комбинация может формироваться в одиночном выходном узле. Таким образом, при условии что количество скрытых элементов довольно велико, для этого вполне достаточно использовать двухслойную сеть. Однако при этом, чтобы сохранить возможность моделирования любых особенностей поведения произвольной функции, активационная функция всегда должна предоставлять базовую нелинейность в той или иной форме. Чтобы разобраться в том, почему это так, заметим, что все одномерные функции могут аппроксимироваться суммой масштабированных/смещенных ступенчатых функций, а большинство функций активации, которые обсуждались в этой главе (например, сигмоида), весьма напоминают ступенчатые функции (см. рис. 1.8). Эта базовая идея выражает суть универсальной теоремы об аппроксимации функций с помощью нейронных сетей. В действительности доказательство способности функций сжатия аппроксимировать любую функцию концептуально аналогично такому же доказательству в отношении гауссовских ядер (по крайней мере на интуитивном уровне). В то же время требуемое количество базовых функций, обеспечивающее высокую точность аппроксимации, в обоих случаях может быть очень большим, тем самым повышая требования, относящиеся к данным, до уровня, делающего их невыполнимыми. По этой причине мелкие сети сталкиваются с извечной проблемой переобучения. Универсальная теорема об аппроксимируемости функций утверждает о возможности удовлетворительной аппроксимации функции, неявно заданной тренировочными данными, но не дает никаких гарантий относительно того, что эта функция будет обобщаться на неизвестные тестовые данные.

1.5.1. Важность нелинейной активации

В предыдущем разделе было предоставлено непосредственное доказательство того факта, что увеличение количества слоев в нейронной сети, в которой используются только линейные функции активации, не дает никакого выигрыша. В качестве примера рассмотрим двухклассовый набор данных, представленный на рис. 1.14 в двух измерениях, обозначенных как x_1 и x_2. Этот набор данных включает два экземпляра, A и B, с координатами (1, 1) и (−1, 1) соответственно, принадлежащих к классу, обозначенному символом $*$. Также существует единственный экземпляр B с координатами (0, 1), принадлежащий к классу, обозначенному символом +. Нейронная сеть, использующая только линейные активации, ни при каких условиях не сможет идеально классифицировать тренировочные данные, поскольку между точками, относящимися к разным классам, невозможно провести разделяющую их прямую линию.

С другой стороны, рассмотрим ситуацию, в которой скрытые элементы используют активацию ReLU и обучаются двум новым признакам h_1 и h_2 следующим образом:

$$h_1 = \max\{x_1,\ 0\},$$
$$h_2 = \max\{-x_1,\ 0\}.$$

Обратите внимание на то, что обе эти цели могут быть достигнуты за счет использования подходящих весов связей, соединяющих вход со скрытым слоем, и применения активации ReLU. Последняя из этих мер устанавливает порог, обрезающий отрицательные значения до нуля. Соответствующие веса связей указаны на рис. 1.14.

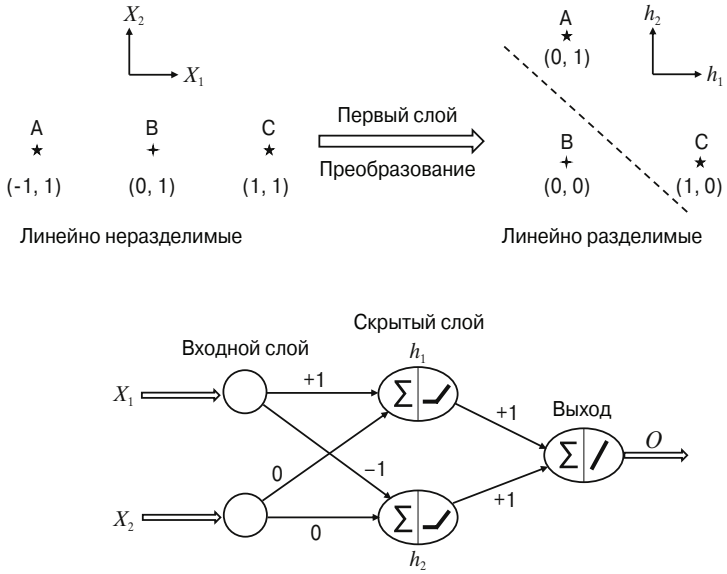


Рис. 1.14. Преобразование линейно неразделимого набора данных в линейно разделимый с помощью нелинейных функций активации

Мы отобразили те же данные на том же рисунке, используя координаты h_1 и h_2. Координаты трех точек в двухмерном скрытом слое равны {(1, 0), (0, 1), (0, 0)}. Это сразу делает очевидным тот факт, что в терминах нового скрытого представления два класса становятся линейно разделимыми. В определенном смысле задача первого слоя заключалась в том, чтобы обеспечить *обучение представлению*, позволяющему получить решение с помощью линейного классификатора. Поэтому, если мы добавим в нейронную сеть единственный линейный выходной слой, то он будет идеально классифицировать наши тренировочные примеры. *Функции активации выполняют нелинейное преобразование,*

обеспечивающее отображение точек данных в линейно разделимый набор. Фактически, если установить веса обеих связей, соединяющих скрытый слой с выходным, равными 1, и применить линейную функцию активации, то выход O будет определяться следующей формулой:

$$O = h_1 + h_2. \tag{1.35}$$

Эта простая линейная функция разделяет два класса, поскольку она всегда принимает значение 1 для обеих точек, обозначенных символом *, и значение 0 для точки, обозначенной символом +. Поэтому мощь нейронных сетей в значительной степени заключается в использовании функций активации. *Обучение весов*, приведенных на рис. 1.14, должно осуществляться в ходе процесса, *управляемого данными*, хотя существует много других возможных вариантов выбора весов, способных сделать скрытое представление линейно разделимым. Поэтому веса, полученные в результате фактического выполнения тренировки, могут отличаться от тех, которые приведены на рис. 1.14. Тем не менее в случае перцептрона нельзя надеяться на существование таких весов, которые обеспечили бы идеальную классификацию этого тренировочного набора данных, поскольку он не разделяется линейно в исходном пространстве. Другими словами, функции активации делают возможными такие нелинейные преобразования данных, мощность которых увеличивается с увеличением количества слоев. Последовательность нелинейных преобразований диктует определенный тип структуры обучаемой модели, мощность которой увеличивается с увеличением глубины последовательности (т.е. количества слоев нейронной сети).

Другим классическим примером является функция *XOR*, в которой две точки {(0, 0), (1, 1)} относятся к одному классу, а другие две точки {(1, 0), (0, 1)} — к другому. Для разделения этих двух классов также можно использовать активацию ReLU, хотя в этом случае потребуются нейроны смещения (см. уравнение 1). Функция *XOR* обсуждается в оригинальной статье, посвященной обратному распространению ошибки [409], поскольку она была одним из факторов, стимулирующих разработку многослойных сетей и способов их тренировки. Функция *XOR* считается лакмусовой бумажкой, позволяющей определить принципиальную возможность предсказания линейно неразделяемых классов с помощью конкретного семейства нейронных сетей. Несмотря на то что выше для простоты использовалась функция активации ReLU, для достижения тех же целей можно использовать большинство других активационных функций.

1.5.2. Снижение требований к параметрам с помощью глубины

Основой глубокого обучения служит идея о том, что повторная композиция функций часто может снижать требования к количеству базовых функций

(вычислительных элементов), обеспечивая его экспоненциальное уменьшение с увеличением количества слоев в сети. Поэтому, даже если количество слоев в сети увеличивается, количество параметров, необходимых для аппроксимации той же функции, резко уменьшается. В результате увеличивается и обобщающая способность сети.

Идея углубления архитектур основана на тех соображениях, что это обеспечивает более эффективное использование повторяющихся закономерностей в распределении данных для уменьшения количества вычислительных элементов, а значит, и для обобщения результатов обучения даже на области пространства данных, не охватываемые предоставленными примерами. Во многих случаях нейронные сети обучаются этим закономерностям путем использования весов в качестве базисных векторов иерархических признаков. И хотя подробное доказательство [340] этого факта выходит за рамки книги, мы предоставим пример, проливающий определенный свет на эту точку зрения. Рассмотрим ситуацию, в которой одномерная функция определяется с помощью 1024 повторяющихся ступеней, имеющих одну и ту же ширину и высоту. Мелкой сети с одним скрытым слоем и ступенчатой функцией активации для моделирования этой функции потребуется 1024 элемента. Однако многослойная сеть будет моделировать 1 ступень в первом слое, 2 ступени в следующем, 4 ступени в третьем и 2^r ступеней в r-м слое (рис. 1.15). Обратите внимание на то, что простейшим признаком является шаблон в виде одной ступени, поскольку он повторяется 1024 раза, тогда как шаблон в виде двух ступеней — более сложный. Поэтому признаки (и изучаемые сетью функции) в последовательных слоях иерархически связаны между собой. В данном случае для моделирования присоединения этих двух шаблонов из предыдущего слоя потребуется в общей сложности 10 слоев и небольшое количество постоянных узлов в каждом слое.

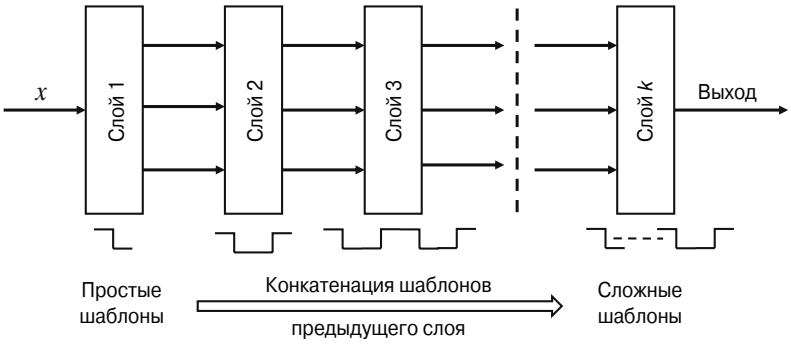


Рис. 1.15. Более глубокие сети могут обучаться более сложным функциям за счет композиции функций, которым были обучены предыдущие слои

Еще один способ, позволяющий подойти к этому вопросу с другой стороны, заключается в следующем. Рассмотрим одномерную функцию, которая принимает одно из значений, 1 и −1, в чередующихся интервалах, и это значение переключается 1024 раза через регулярные интервалы аргумента. Единственная возможность имитировать такую функцию с помощью ступенчатых функций активации (содержащих лишь один переключатель в значении) — это использовать 1024 таких функции (или небольшое кратное от этого количества). Однако в нейронной сети с 10 скрытыми слоями и всего лишь 2 элементами в каждом слое существует $2^{10} = 1024$ пути от источника к выходу. Если функция, которой должна обучиться сеть, обладает той или иной регулярностью, то часто существует возможность обучить параметры слоев таким образом, чтобы эти 1024 пути смогли вобрать в функцию всю сложность 1024 различных переключателей значений. Ранние слои обучаются более детальным шаблонам, более поздние — более высокоуровневым шаблонам. Поэтому общее количество требуемых узлов *на порядок меньше* того, которое потребовалось бы для однослойной сети. Это означает, что объем необходимых для обучения данных также на порядок меньше. Причина заключается в том, что многослойная сеть неявно *ищет повторяющиеся регулярности и обучается им* на меньшем объеме данных, чем если бы она пыталась явно обучаться каждому повороту и изгибу целевой функции. Такое поведение становится интуитивно очевидным при использовании сверточных нейронных сетей для обработки изображений, когда начальные слои моделируют простые признаки наподобие линий, средний слой — элементарные формы, а последующий — сложные формы наподобие лица. С другой стороны, одиночный слой испытывал бы трудности при моделировании каждой черты лица. Это наделяет более глубокую модель улучшенной способностью к обобщению, а также способностью к обучению на меньших объемах данных.

Однако увеличение глубины сети не лишено недостатков. Более глубокие сети часто труднее поддаются обучению и подвержены всем типам нестабильного поведения, в том числе проблемам затухающих и взрывных градиентов. Глубокие сети также проявляют явную нестабильность в отношении выбора параметров. Эти проблемы чаще всего удается устранить за счет тщательного продумывания функций, вычисляемых в узлах, а также использования процедур предварительного обучения для повышения производительности сети.

1.5.3. Нестандартные архитектуры нейронных сетей

Выше был дан обзор наиболее распространенных способов конструирования и организации типичных нейронных сетей. Однако существует много вариаций этих способов, которые обсуждаются ниже.

1.5.3.1. Размытие различий между входными, скрытыми и выходными слоями

Вообще говоря, при обсуждении структуры нейронных сетей основное внимание уделяется главным образом сетям прямого распространения с послойной организацией и последовательным расположением входных, скрытых и выходных слоев. Другими словами, все входные узлы передают данные первому скрытому слою, скрытые слои последовательно передают данные друг другу, а последний скрытый слой передает данные выходному слою. Вычислительные элементы часто определяют как *функции сжатия* (squashing functions), применяемые к линейным комбинациям входа. Обычно скрытый слой не принимает входные данные, а функция потерь обычно вычисляется без учеты значений в скрытых слоях. Из-за этого смещения акцентов легко забыть о том, что нейронная сеть может быть определена в виде *параметризованного вычислительного графа любого типа*, не требующего этих ограничений для того, чтобы работал механизм обратного распространения ошибки. Вообще говоря, вполне возможно, чтобы промежуточные слои включали входы или принимали участие в вычислении функции потерь, хотя такие подходы менее распространены. Например, была предложена [515] сеть, идея которой была подсказана *случайными лесами* [49] и которая допускает наличие входов в разных слоях. Один из примеров такой сети приведен на рис. 1.16. В данном случае размытие различий между входными и скрытыми слоями становится очевидным.

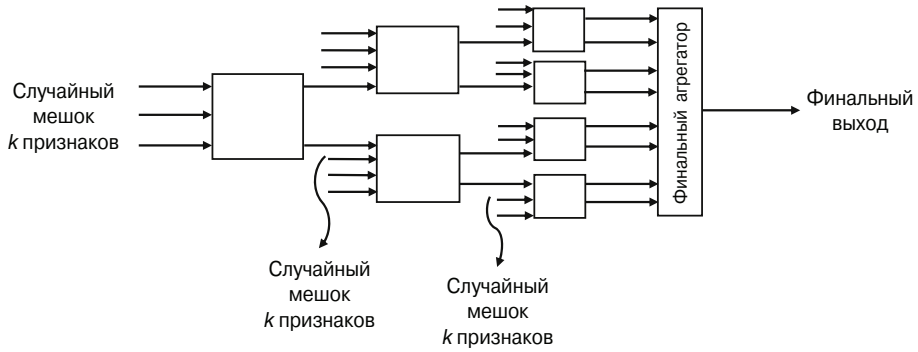


Рис. 1.16. Пример нестандартной архитектуры, в которой входы передают данные слоям, отличным от первого скрытого слоя. При условии что нейронная сеть не содержит циклов (или может быть преобразована в ациклическое представление), веса базового вычислительного графа могут обучаться с использованием динамического программирования (обратного распространения ошибки)

В других вариациях базовой архитектуры прямого распространения функции потерь вычисляются не только в выходных узлах, но и в скрытых. Вклады в скрытых узлах часто принимают форму *штрафов*, действующих в качестве ре-

гуляризаторов. Например, методы этого типа используются для обучения разреженным признакам посредством штрафования скрытых узлов (главы 2 и 4). В данном случае размываются различия между скрытыми и выходными слоями.

В качестве еще одного примера предложенных недавно вариантов проектирования нейронных сетей можно привести архитектуру с *замыкающими соединениями* (skip connections) [184], в которой входы одного слоя могут напрямую соединяться с другими слоями, расположенными далее слоя, непосредственно следующего за данным. Такой подход порождает по-настоящему глубокие модели. Например, 152-слойная архитектура, получившая название *ResNet* [184], обеспечила эффективность работы сети на уровне человека в задачах распознавания образов. И хотя эта архитектура не размывает различия между входными, скрытыми и выходными словами, ее структура отличается от структуры традиционных сетей прямого распространения, в которых допустимы только соединения между смежными последовательными слоями. В сетях этого типа используется итеративный подход к *конструированию признаков* (feature engineering) [161], в соответствии с которым признаки в поздних слоях представляют собой итеративные уточнения признаков в ранних слоях. В противоположность этому традиционный подход к конструированию признаков — иерархический, т.е. признаки в более поздних слоях представляют собой все более абстрактные представления признаков из предыдущих слоев.

1.5.3.2. Необычные операции и сети сумм и произведений

Некоторые нейронные сети наподобие сетей с ячейками *долгой краткосрочной памяти* (long short-term memory — LSTM) и сверточных нейронных сетей определяют различные типы операций мультипликативного “забывания”, свертки и пулинга, выполняемые над переменными, существующими не строго в тех формах, которые обсуждались в этой главе. В настоящее время такие архитектуры настолько интенсивно используются при обработке текста и изображения, что они больше не рассматриваются как необычные.

Еще одним уникальным типом архитектуры является *сеть сумм и произведений* (sum-product network) [383]. В данном случае узлы являются либо узлами суммирования, либо узлами умножения. Узлы суммирования аналогичны традиционным линейным преобразованиям, выполняемым над набором взвешенных ребер вычислительных графов. Однако веса могут иметь только положительные значения. Узлы умножения просто умножают свои входы без использования весов. Следует отметить, что существует несколько вариаций этих узлов, различающихся способом вычисления произведения. Например, если входы — это два скаляра, то можно вычислить их произведение. Если входы — это два вектора равной длины, то можно вычислить их поэлементное произведение. Некоторые библиотеки глубокого обучения поддерживают вычисление произведений такого типа. Вполне естественно, что для максимизации

выразительной способности сети слои суммирования могут чередоваться со слоями умножения.

Сети сумм и произведений довольно выразительны, и часто возможно создание их глубоких вариаций с высокой степенью выразительности [30, 93]. Очень важно то, что любая математическая функция может быть аппроксимирована полиномиальной функцией своих входов. Поэтому почти любая функция может быть выражена с помощью архитектуры сумм и произведений, хотя более глубокие архитектуры допускают моделирование с использованием более развитых структур. В отличие от традиционных нейронных сетей, в которых нелинейность встраивается с помощью функций активации, в сетях сумм и произведений ключом к нелинейности является операция умножения.

Проблемы обучения

Используя различные типы вычислительных операций в узлах, часто целесообразно проявлять гибкость и выходить за рамки известных преобразований и функций активации. Кроме того, соединения между узлами не обязательно должны структурироваться послойно, а узлы в скрытых слоях могут включаться в вычисление функции потерь. Если базовый вычислительный граф является ациклическим, то можно легко обобщить алгоритм обратного распространения ошибки на любой тип архитектуры и любые вычислительные операции. В конце концов, алгоритм динамического программирования (наподобие обратного распространения ошибки) можно использовать практически на любом типе направленного ациклического графа, в котором для инициализации динамической рекурсии можно использовать несколько узлов. Очень важно иметь в виду, что архитектуры, спроектированные на основе понимания особенностей конкретной предметной области, часто могут обеспечивать результаты, превосходящие результаты, полученные методами черного ящика, в которых используются полносвязные сети прямого распространения.

1.6. Распространенные архитектуры нейронных сетей

Существует несколько типов нейронных архитектур, применяемых в задачах машинного обучения. В этом разделе дается краткий обзор архитектур, которые будут более подробно обсуждаться в следующих главах.

1.6.1. Имитация базового машинного обучения с помощью мелких моделей

Большинство базовых моделей машинного обучения, таких как линейная регрессия, классификация, метод опорных векторов, логистическая регрессия,

сингулярное разложение и матричная факторизация, можно имитировать с помощью мелких нейронных сетей, содержащих не более одного-двух слоев. Базовые архитектуры такого типа заслуживают изучения, поскольку это будет косвенной демонстрацией возможностей нейронных сетей. Большую часть того, что мы знаем о машинном обучении, можно сымитировать с помощью относительно простых моделей! Кроме того, многие базовые модели нейронных сетей наподобие *модели обучения Уидроу — Хоффа* (Widrow–Hoff learning model) самым непосредственным образом связаны с традиционными моделями машинного обучения, такими как дискриминант Фишера, хотя они и были предложены независимо. Следует отметить один важный момент: глубокие архитектуры часто создаются путем продуманного образования стеков более простых моделей. Нейронные модели для базовых моделей машинного обучения обсуждаются в главе 2. Там же будут рассматриваться приложения для работы с текстом и рекомендательные системы.

1.6.2. Сети радиально-базисных функций

Сети *радиально-базисных функций* (radial basis function — RBF) представляют одну из забытых архитектур, которыми богата история нейронных сетей. Эти архитектуры редко встречаются в наши дни, хотя и обладают значительным потенциалом применительно к некоторым типам задач. Одним из лимитирующих факторов является то, что они не обладают глубиной и, как правило, насчитывают всего лишь два слоя. Первый строй конструируется с расчетом на самообучение, тогда как второй тренируется с использованием методов обучения с учителем. Эти сети фундаментально отличаются от сетей прямого распространения, и источник их возможностей кроется в большом количестве узлов в слое обучения без учителя. Базовые принципы использования RBF-сетей сильно отличаются от принципов использования сетей прямого распространения в том смысле, что усиление их возможностей достигается за счет увеличения размера пространства признаков, а не глубины. Этот подход базируется на *теореме Ковера о разделимости образов (шаблонов)* [84], которая утверждает, что задачи классификации образов линеаризуются легче после перевода их в многомерное пространство с помощью нелинейного преобразования. Каждый узел второго слоя сети содержит прототип, и активация определяется на основе сходства входных данных и прототипа. Затем эти активации комбинируются с обученными весами следующего слоя для создания окончательного прогноза. Такой подход очень напоминает классификацию по методу ближайших соседей, за исключением того, что веса во втором слое обеспечивают дополнительный уровень управления процессом обучения. Другими словами, данный подход представляет собой *контролируемый метод ближайших соседей* (supervised nearest-neighbor method).

В частности, известно, что реализации метода опорных векторов представляют собой контролируемые варианты классификаторов по методу ближайших соседей, в которых ядерные функции комбинируется с обучаемыми весами для взвешивания соседних точек в окончательном прогнозе [6]. Сети радиальных базисных функций могут использоваться для имитации ядерных методов, таких как метод опорных векторов. В некоторых типах задач, таких как классификация, эти архитектуры могут работать эффективнее готовых ядер. Причина заключается в большей общности этих моделей, что предоставляет больше возможностей для экспериментов. Кроме того, иногда удается получить дополнительный выигрыш за счет увеличенной глубины обучаемых слоев. Полный потенциал сетей радиальных базисных функций все еще остается не до конца исследованным в литературе, поскольку эта архитектура была в значительной степени забыта, и исследователи сосредоточили основное внимание на методах прямого распространения. Сети радиально-базисных функций обсуждаются в главе 5.

1.6.3. Ограниченные машины Больцмана

В ограниченных машинах Больцмана (restricted Boltzmann machine — RBM) для создания архитектур нейронных сетей, предназначенных для моделирования данных на основе самообучения, используется понятие энергии, которая подлежит минимизации. Эти методы особенно полезны при создании генеративных моделей и тесно связаны с вероятностными графическими моделями [251]. Ограниченные машины Больцмана берут свое начало в *сетях Хопфилда* [207], которые могут использоваться для сохранения ранее полученной информации. Результатом обобщения стохастических вариантов этих сетей явились *машины Больцмана*, в которых скрытые слои моделировали генеративные аспекты данных.

Ограниченные машины Больцмана часто применяются в обучении без учителя и для снижения размерности данных, хотя они также могут использоваться для обучения с учителем. В то же время, поскольку они не были изначально приспособлены для обучения с учителем, тренировке с учителем часто предшествовала фаза тренировки без учителя. Это естественным образом привело к идее предварительного обучения, которая оказалась чрезвычайно плодотворной для обучения с учителем. RBM были одними из первых моделей, которые начали применяться в глубоком обучении, особенно без учителя.

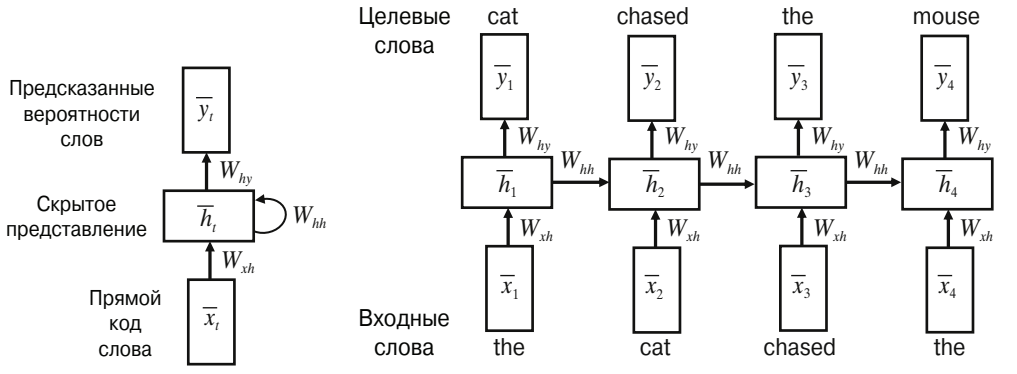
В конечном счете подход, основанный на предварительном обучении, был перенесен на другие типы моделей. Поэтому архитектура RBM имеет еще и историческое значение в том смысле, что она инициировала разработку некоторых методологий тренировки для глубоких моделей. Процесс тренировки ограниченной машины Больцмана заметно отличается от процесса тренировки сетей

прямого распространения. В частности, эти модели нельзя тренировать с использованием механизма обратного распространения ошибки, и для их тренировки необходимо прибегать к семплированию данных по методу Монте-Карло. Обычно для тренировки RBM используют конкретный алгоритм, а именно так называемый *алгоритм контрастивной дивергенции* (contrastive divergence algorithm). Ограниченные машины Больцмана обсуждаются в главе 6.

1.6.4. Рекуррентные нейронные сети

Рекуррентные нейронные сети предназначены для работы с последовательными данными, такими как текстовые цепочки, временны*е* ряды и другие дискретные последовательности. В таких случаях вход имеет форму $\overline{x}_1 \ldots \overline{x}_n$, где $\overline{x}_t$ — d-мерная точка в момент времени t. Например, вектор x_t может содержать d значений, соответствующих t-му отсчету (моменту времени t) многомерного ряда (с d различными рядами). При работе с текстом вектор $\overline{x}_t$ будет содержать закодированное с помощью *прямого кодирования* слово, соответствующее t-му отсчету. При прямом кодировании мы имеем вектор, длина которого равна размеру словаря и компонента которого, соответствующая данному слову, равна 1. Все остальные компоненты равны нулю.

В случае последовательностей важно то, что последовательные слова зависят друг от друга. Поэтому целесообразно получать конкретный вход x_t лишь *после* того, как более равные входы уже были получены и преобразованы в скрытое состояние. Традиционный тип сетей прямого распространения, в котором все входы передаются первому слою, не обеспечивает достижения этой цели. Поэтому рекуррентная нейронная сеть позволяет входу $\overline{x}_t$ непосредственно взаимодействовать со скрытым состоянием, созданным из входов, соответствующих предыдущим временны*м* отсчетам. Базовая архитектура рекуррентной нейронной сети показана на рис. 1.17, *а*. Ключевой аспект заключается в том, что в каждый момент времени существует вход $\overline{x}_t$ и скрытое состояние $\overline{h}_t$, которое изменяется, как только поступает новая точка данных. Для каждого отсчета также имеется выходное значение $\overline{y}_t$. Например, в случае временно*г*о ряда выходом $\overline{y}_t$ может быть прогнозируемое значение $\overline{x}_{t+1}$. Если речь идет о работе с текстом, когда предсказывается следующее слово, то такой подход называют *моделированием языка* (language modeling). В некоторых приложениях мы выводим значения $\overline{y}_t$ не для каждого отсчета, а лишь в конце последовательности. Например, если мы пытаемся классифицировать тональность предложения как “положительная” или “отрицательная”, то выходное значение будет выведено только для последнего отсчета.



а) Рекуррентная нейронная сеть б) Представление *а*, развернутое во времени

Рис. 1.17. Рекуррентная нейронная сеть и ее развернутое во времени представление

Скрытое состояние в момент времени t определяется функцией входного вектора в момент времени t и скрытого вектора в момент времени $(t-1)$:

$$\overline{h}_t = f(\overline{h}_{t-1}, \overline{x}_t). \tag{1.36}$$

Для обучения выходным вероятностям на основе скрытых состояний используется отдельная функция $\overline{y}_t = g(\overline{h}_t)$. Обратите внимание на то, что функции $f(\cdot)$ и $g(\cdot)$ остаются одними и теми же для каждого отсчета. Здесь скрыто неявное предположение о том, что временны*е* ряды демонстрируют некоторый уровень *стационарности*, т.е. базовые свойства не меняются с течением времени. Несмотря на то что в реальных задачах это условие не выполняется со всей строгостью, такое предположение достаточно разумно для того, чтобы использовать его в целях регуляризации.

В данном случае ключевую роль играет изображенный на рис. 1.17, *а*, цикл, наличие которого приводит к тому, что скрытое состояние нейронной сети изменяется каждый раз при появлении на входе нового значения $\overline{x}_t$. На практике мы имеем дело с последовательностями конечной длины, и поэтому имеет смысл развернуть этот цикл в сеть с временны*ми* слоями, которая выглядит как сеть прямого распространения (рис. 1.17, *б*). Заметьте, что в данном случае у нас имеются отдельные узлы для каждого скрытого состояния, соответствующего отдельной временно*й* метке, и цикл был развернут в сеть прямого распространения. Это представление математически эквивалентно тому, которое приведено на рис. 1.17, *а*, но оно гораздо понятнее в силу его сходства с традиционной сетью. Обратите внимание на то, что, в отличие от сетей прямого распространения, в этой развернутой сети входы встречаются также в промежуточных слоях. Матрицы весов соединений *совместно используются многими соединениями* в развернутой во времени сети, тем самым гарантируя, что в каждый момент вре-

мени применяется одна и та же функция. Совместное использование весов играет ключевую роль в обучении сети специфическим для конкретных данных внутренним закономерностям. Алгоритм обратного распространения ошибки учитывает это и временну*ю* длину при обновлении весов в процессе обучения. Этот специальный тип обратного распространения ошибки называют *обратным распространением во времени* (backpropagation through time — BPTT). В силу рекурсивной природы уравнения 1.36 рекуррентная сеть обладает *способностью вычислять функцию входов переменной длины*. Другими словами, рекурсию уравнения 1.36 можно развернуть, чтобы определить функцию для $\overline{h}_t$ в терминах t входов. Например, начиная с члена $\overline{h}_0$, значение которого обычно фиксируется равным некоторому постоянному вектору, мы имеем $\overline{h}_1 = f(\overline{h}_0, \overline{x}_1)$ и $\overline{h}_2 = f(f(\overline{h}_0, \overline{x}_1), \overline{x}_2)$. Обратите внимание на то, что $\overline{h}_1$ — это функция только $\overline{x}_1$, тогда как $\overline{h}_2$ — функция как $\overline{x}_1$, так и $\overline{x}_2$. Поскольку выход $\overline{y}_t$ — функция $\overline{h}_t$, то эти свойства наследуются также и $\overline{y}_t$. В общем случае мы можем записать следующее соотношение:

$$\overline{y}_t = F_t\left(\overline{x}_1, \overline{x}_2, \ \ldots, \overline{x}_t\right). \tag{1.37}$$

Заметьте, что функция $F_t(\cdot)$ меняется с изменением значения t. Такой подход особенно полезен в случае входов переменной длины наподобие текстовых предложений. Более подробно о рекуррентных нейронных сетях речь пойдет в главе 7, и там же будет обсуждаться применение рекуррентных нейронных сетей в различных областях.

Интересным теоретическим свойством рекуррентных нейронных сетей является их *полнота по Тьюрингу* [444]. Это означает, что при наличии достаточно большого количества данных и вычислительных ресурсов рекуррентная нейронная сеть может имитировать любой алгоритм. Однако на практике это теоретическое свойство не приносит никакой пользы ввиду значительных практических проблем с обобщаемостью рекуррентных сетей на длинные последовательности. Требуемые для этого объем данных и размер скрытых состояний возрастают с увеличением длины последовательности настолько быстро, что их практическая реализация становится невозможной. Кроме того, возникают практические трудности с выбором оптимальных параметров, обусловленные проблемами затухающих и взрывных градиентов. Вследствие этого были предложены специальные варианты рекуррентных нейронных сетей, например такие, в которых используется долгая краткосрочная память. Эти более сложные архитектуры также обсуждаются в главе 7. Кроме того, с помощью усовершенствованных вариантов рекуррентной архитектуры, таких как нейронные машины Тьюринга, в некоторых приложениях удалось получить лучшие результаты, чем с помощью рекуррентных нейронных сетей.

1.6.5. Сверточные нейронные сети

Сверточные нейронные сети, идея которых была почерпнута из биологии, применяются в компьютерном зрении для классификации изображений и обнаружения объектов. В основу этой идеи легли результаты Хьюбела и Визеля [212], исследовавших работу зрительной коры головного мозга кошек, в соответствии с которыми определенные участки зрительного поля возбуждают определенные нейроны. Этот широкий принцип был использован при проектировании разреженной архитектуры сверточных нейронных сетей. Первой базовой архитектурой, основанной на этой биологической модели, был *неокогнитрон*, впоследствии обобщенный до архитектуры *LeNet-5* [279]. В архитектуре сверточной нейронной сети каждый слой является 3-мерным и имеет пространственную протяженность и глубину, которая соответствует количеству признаков. Понятие "глубины" слоя сверточной нейронной сети отличается[4] от понятия глубины в смысле количества слоев. Во входном слое упомянутые признаки соответствуют цветовым каналам типа RGB (т.е. красному, зеленому и синему цветам), тогда как в скрытых каналах эти признаки представляют карты скрытых признаков, кодирующие различные типы форм в изображении. Если входы задаются в градациях серого (как в LeNet-5), то входной слой будет иметь глубину 1, но последующие слои по-прежнему будут трехмерными. Такая архитектура содержит слои двух типов, которые называют *сверточным слоем* (convolution layer) и *слоем субдискретизации* (subsampling layer) соответственно.

Для сверточных слоев определяется операция *свертки* (convolution), в которой для трансляции активаций из одного слоя в следующий используется фильтр. Операция свертки использует трехмерный фильтр весов той же глубины, что и текущий слой, но меньшей пространственной протяженности. Значение скрытого состояния в следующем слое (после применения функции активации наподобие ReLU) определяется скалярным (точечным) произведением между всеми весами фильтра и любой выбранной в слое пространственной областью (того же размера, что и фильтр). Данная операция между фильтром и пространственными областями в слое выполняется в каждой возможной позиции для определения следующего слоя (в котором сохраняются пространственные соотношения между активациями из предыдущего слоя).

Сверточная нейронная сеть характеризуется высокой разреженностью соединений, поскольку любая активация отдельного слоя является функцией только небольшой пространственной области предыдущего слоя. Все слои, за исключением последнего набора из двух-трех слоев, сохраняют свою

[4] Это перегрузка терминологии сверточных нейронных сетей. Смысл слова "глубина" выводится из контекста, в котором оно используется.

пространственную структуру, что обеспечивает возможность пространственной визуализации того, какие именно части изображения влияют на определенные части активаций в слое. Признаки в слоях более низкого уровня захватывают линии и другие примитивные формы, тогда как признаки в слоях более высокого уровня захватывают более сложные формы наподобие замкнутых петель (которые обычно встречаются в начертаниях многих цифр). Поэтому более поздние слои могут создавать цифры путем композиции форм, сохраненных в этих интуитивных признаках. Это классический пример того, как понимание внутренних семантических зависимостей, существующих среди данных определенного типа, используется для проектирования более "умных" архитектур. В дополнение к этому субдискретизирующий слой просто усредняет значения в локальных областях размером 2×2, тем самым снижая размерность слоев по каждому пространственному измерению в 2 раза. Архитектура сети LeNet-5 приведена на рис. 1.18. Ранее сеть LeNet-5 использовалась несколькими банками для распознавания рукописных цифр на чеках.

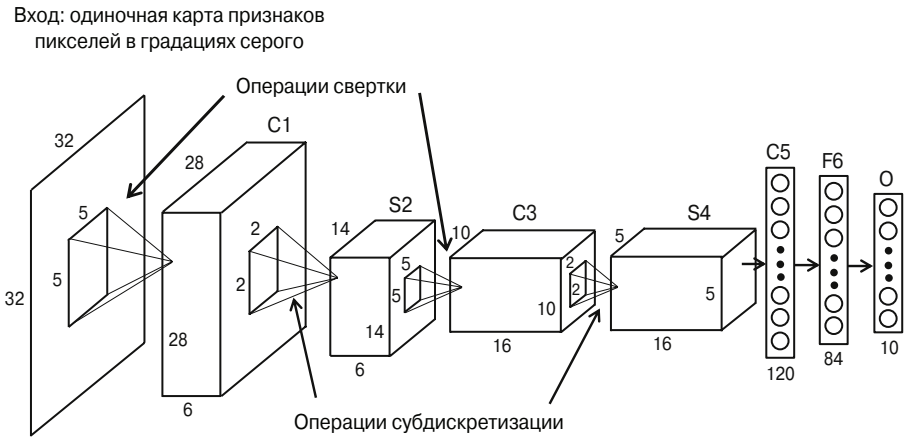


Рис. 1.18. LeNet-5: одна из первых сверточных нейронных сетей

Сверточные нейронные сети исторически оказались наиболее успешными из всех типов нейронных сетей. Они широко применяются для распознавания образов, обнаружения/локализации объектов и даже для обработки текста. При решении задач классификации изображений эти сети недавно продемонстрировали производительность, превосходящую возможности человека [184]. Пример сверточных нейронных сетей хорошо подтверждает тот факт, что варианты архитектурных решений нейронной сети должны выбираться с учетом семантических особенностей данных конкретной предметной области. В частном случае сверточной нейронной сети понимание этих особенностей было достигнуто на основе наблюдений за тем, как работает зрительная кора головного мозга кошек, а также за счет интенсивного использования пространственных

соотношений между пикселями. Кроме того, этот факт служит дополнительным свидетельством того, что дальнейшие достижения нейронауки также могут оказаться полезными для разработки методов искусственного интеллекта.

Сверточные нейронные сети, предварительно обученные на таких публично доступных ресурсах, как ImageNet, часто предлагаются в готовом для применения в других приложениях виде. Это достигается за счет использования в сверточной сети большинства предварительно обученных весов без каких-либо изменений, за исключением последнего классифицирующего слоя. Веса в этом слое обучаются на данных конкретной решаемой задачи. Тренировка последнего слоя необходима по той причине, что метки классов в данной задаче могут отличаться от тех, которые используются в ImageNet. Тем не менее веса в ранних слоях все еще остаются полезными, поскольку они обучены различным типам форм в изображениях, которые могут пригодиться в задачах классификации практически любого типа. Кроме того, активации признаков в предпоследнем слое могут быть использованы в приложениях даже в целях обучения без учителя. Например, можно создать многомерное представление произвольного набора изображений, пропуская каждое изображение через сверточную нейронную сеть и извлекая активации предпоследнего слоя. Впоследствии к этому представлению можно применить любой тип индексирования для изображений, аналогичных предъявленному целевому изображению. Такой подход часто демонстрирует на удивление хорошие результаты при извлечении изображений в силу семантической природы признаков, которым обучается сеть. Следует отметить, что использование предварительно обученных сверточных сетей стало настолько популярным, что их тренировка с нуля выполняется лишь в редких случаях. Сверточные нейронные сети подробно обсуждаются в главе 8.

1.6.6. Иерархическое конструирование признаков и предварительное обучение моделей

Многие углубленные архитектуры, охватывающие архитектуры прямого распространения, включают несколько слоев, в которых последовательные преобразования входов от предыдущего слоя приводят к представлениям данных все более увеличивающейся сложности. Значения в каждом скрытом слое, соответствующие отдельному входу, включают преобразованное представление входной точки данных, содержащее все более полную информацию о целевом значении, которому мы пытаемся обучить сеть, по мере приближения к выходному узлу. Как было показано в разделе 1.5.1, преобразованные подходящим образом представления признаков лучше соответствуют простым типам предсказаний в выходном слое. Это усложнение является результатом применения нелинейных активаций в промежуточных слоях. Ранее в качестве функций активации чаще всего выбирались сигмоида и гиперболический тангенс, однако в последнее

время все бо́льшую популярность завоевывает активация ReLU, использование которой облегчает устранение проблем затухающего и взрывного градиента (раздел 3.4.2). В задачах классификации последний слой может рассматриваться в качестве простого прогнозного слоя, который содержит всего один выходной нейрон в случае регрессии и представляет собой сигмоиду/знаковую функцию в случае бинарной классификации. Более сложные выходы могут требовать использования большего количества узлов. Одна из точек зрения на такое разделение труда между скрытыми слоями и последним прогнозным слоем заключается в том, что ранние слои создают представление признаков, которое более всего соответствует данной задаче. Затем последний слой использует все преимущества этого представления изученных признаков. Подобное разделение труда показано на рис. 1.19. Важно то, что признаки, которым сеть обучилась в скрытых слоях, часто (но не всегда) обобщаются на другие наборы данных и задачи, относящиеся к той же предметной области (например, обработка текста, изображений и т.п.). Этой особенностью можно воспользоваться самыми разными способами, просто заменив выходной узел (узлы) предварительно обученной сети выходным слоем, учитывающим специфику конкретного приложения (например, используя слой линейной регрессии вместо слоя сигмоидной классификации) в отношении данных и решаемой задачи. В дальнейшем обучение новым данным может потребоваться лишь для весов этого нового замещающего слоя при фиксированных весах остальных слоев.

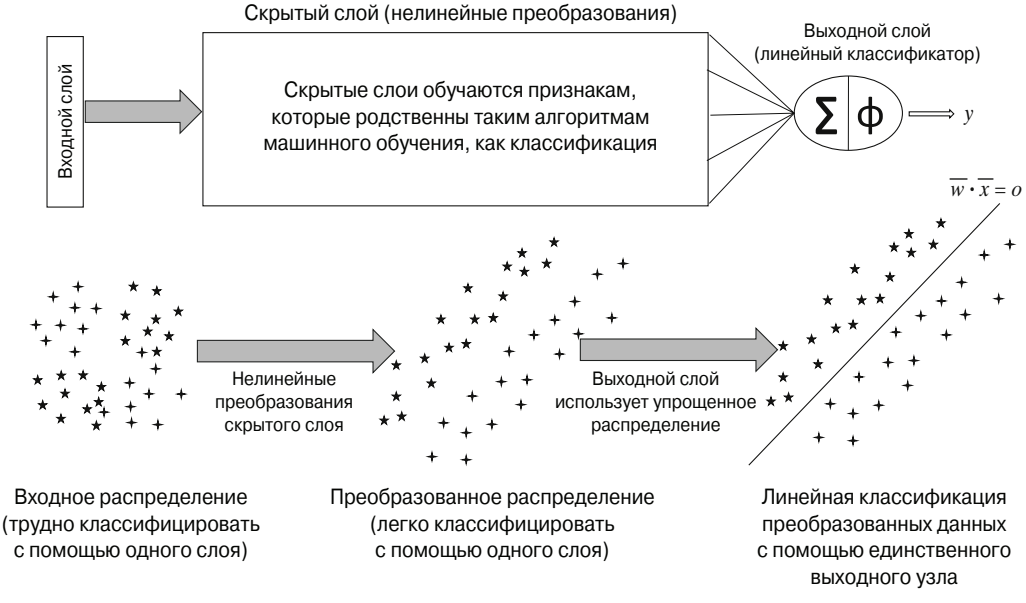


Рис. 1.19. Роль скрытых слоев в конструировании признаков

Выходом каждого скрытого слоя является преобразованное представление признаков данных, размерность которого определяется количеством элементов в данном слое. Этот процесс можно рассматривать в качестве своеобразного процесса построения иерархии признаков, когда признаки в ранних слоях представляют примитивные характеристики данных, в то время как признаки в более поздних слоях представляют сложные характеристики с семантической значимостью для меток классов. Данные, представленные в терминах признаков в более поздних слоях, часто ведут себя лучше (например, являются линейно разделимыми) в силу того, что они отражают семантическую природу признаков, изученных путем преобразования. В некоторых задачах, например в случае сверточных нейронных сетей для обработки изображений, такой тип поведения проявляется в наглядной, визуально интерпретируемой форме. В сверточных нейронных сетях признаки в ранних слоях захватывают в наборе изображений детальные, но примитивные формы наподобие линий или границ однородности изображений. С другой стороны, признаки в более поздних слоях захватывают более сложные формы наподобие шестиугольников, сот и т.п., в зависимости от типа изображений, предоставленных в качестве тренировочных данных. Обратите внимание на то, что такие семантически интерпретируемые формы зачастую тесно коррелируют с метками классов в задачах обработки изображений. Например, почти любое изображение будет содержать линии или границы, тогда как для изображений, принадлежащих к некоторым определенным классам, более характерным будет наличие в них шестиугольников или сот. Эта особенность упрощает задачу классификации представлений более поздних слоев с помощью простых моделей наподобие линейных классификаторов. Признаки, полученные в более ранних слоях, используются в качестве строительных кирпичиков для создания более сложных признаков. Этот общий принцип “сведения воедино” простых признаков для создания более сложных признаков лежит в основе успехов, достигнутых с помощью нейронных сетей. Использование этого способа тщательно продуманными способами оказалось полезным и для предварительного обучения моделей. Практику использования предварительно обученных моделей также называют *переносимым обучением* (transfer learning).

Суть одного специфического типа переносимого обучения, обычно применяемого в нейронных сетях, заключается в том, что данные и структура, доступные для некоторого набора данных, используются с целью обучения признакам, относящимся к данной предметной области в целом. Классическим примером могут служить текстовые или графические данные. В случае текстовых данных представления слов создаются с использованием стандартизированных эталонных наборов данных, таких как *Википедия* [594], и моделей наподобие *word2vec*. Такие данные и модели могут быть использованы практически в любом приложении для обработки текста, поскольку природа

текстовых данных не изменяется существенно при переходе от одного приложения к другому. Аналогичный подход часто применяется и в отношении изображений, когда *набор данных ImageNet* (раздел 1.8.2) используется для предварительного обучения сверточных нейронных сетей и предоставляет готовые к использованию признаки. Любой набор изображений можно преобразовать в многомерное представление, загрузив предварительно обученную сверточную нейронную сеть и пропустив через нее изображения. Кроме того, при наличии дополнительных данных, специфических для приложения, можно регулировать уровень переносимого обучения в зависимости от объема имеющихся данных. Это достигается за счет тонкой настройки подмножества слоев в предварительно обученной сети с помощью этих дополнительных данных. Если доступен лишь небольшой объем данных, специфических для приложения, то можно зафиксировать веса ранних слоев на их предварительно обученных значениях и настроить лишь последние несколько слоев нейронной сети. Ранние слои часто содержат примитивные признаки, которые легче обобщаются на произвольные приложения. Например, в сверточной нейронной сети ранние слои обучаются таким примитивным признакам, как границы областей изображения с однородными характеристиками, которые встречаются в самых разных изображениях, от моркови до грузовика. С другой стороны, поздние слои содержат сложные признаки, которые могут зависеть от особенностей коллекции изображений (например, колесо грузовика или срез верхушки моркови). В подобных случаях имеет смысл настраивать только последние слои. Если же объем доступных данных, специфических для приложения, достаточно велик, то можно настраивать большее количество слоев. Поэтому глубокие сети обеспечивают значительную гибкость в отношении способов реализации переносимого обучения с использованием предварительно обученных моделей нейронных сетей.

1.7. Дополнительные темы

Ряд тем глубокого обучения вызывает постоянно возрастающий интерес, и в этих областях недавно были достигнуты заметные успехи. Несмотря на то что некоторые из методов ограничены доступными в настоящее время вычислительными возможностями, они обладают значительным потенциалом.

1.7.1. Обучение с подкреплением

В обычных приложениях искусственного интеллекта нейронные сети должны обучаться тому, чтобы предпринимать те или иные действия в постоянно изменяющихся и динамически развивающихся ситуациях. Примерами могут служить обучающиеся роботы и беспилотные автомобили. Во всех подобных случаях критическое значение имеет предположение о том, что обучаемой

системе ничего заранее не известно о том, какая последовательность действий будет наиболее подходящей, и все обучение, так называемое *обучение с подкреплением* (reinforcement learning), проводится посредством присуждения вознаграждения за правильность совершаемых системой действий. Эти типы обучения соответствуют динамической последовательности действий, которую трудно моделировать, используя традиционные методы машинного обучения. В данном случае ключевым является предположение о том, что такие системы слишком сложны для явного моделирования, но достаточно просты для того, чтобы за каждое правильное действие обучаемой системы ей можно было назначить вознаграждение.

Представьте ситуацию, когда требуется с нуля обучить систему видеоигре, правила которой ей не известны заранее. Видеоигры представляют собой великолепную тестовую площадку для методов обучения с подкреплением. Как и в реальных ситуациях, количество возможных *состояний* (т.е. уникальных позиций в игре) может быть настолько большим, что уже один его подсчет становится нереальным, и в этих условиях выбор оптимального хода критически зависит от знания того, какая информация о конкретном состоянии игры действительно важна для модели. Кроме того, поскольку невозможно вести игру, вообще не зная правил, обучаемая система должна собрать необходимые сведения, совершая какие-то действия, как это делает мышь, исследуя лабиринт, чтобы изучить его структуру. Поэтому собранные данные испытывают значительное смещение в результате пользовательских действий, что создает неблагоприятные условия для обучения. Успешная тренировка методов обучения с подкреплением — вот путь, ведущий к *самообучающимся* системам, Святому Граалю искусственного интеллекта. И хотя поле обучения с подкреплением было исследовано независимо от поля нейронных сетей, их взаимодополняемость свела их вместе. Методы глубокого обучения могут быть полезными для обучения представлениям признаков на многомерных сенсорных входах (например, это могут быть пиксели видеоигры или пиксели поля “зрения” робота). Кроме того, методы обучения с подкреплением часто применяются для поддержки различных типов алгоритмов нейронных сетей, например *механизмов внимания*. Методы обучения с подкреплением обсуждаются в главе 9.

1.7.2. Отделение хранения данных от вычислений

Важным аспектом нейронных сетей является тесная интеграция хранения данных и вычислений. Например, состояния нейронной сети могут рассматриваться как некоторый тип временной памяти, во многом напоминающий своим поведением регистры центрального процессора компьютера. Но что если мы хотим построить нейронную сеть, в которой можно контролировать, откуда читать данные и куда их записывать? Эта цель достигается за счет использования

внимания и *внешней памяти*. Механизмы внимания могут применяться в различных задачах, таких как обработка изображений, где внимание последовательно фокусируется на небольших участках изображения. Эти методики также используются в машинном переводе. Нейронные сети, имеющие возможность строго контролировать доступ по чтению и записи к внешней памяти, называют *нейронными машинами Тьюринга* (neural Turing machines) [158] или *сетями памяти* (memory networks) [528]. И хотя они являются не более чем усовершенствованными вариантами рекуррентных нейронных сетей, они демонстрируют значительное превосходство над своими предшественниками в отношении круга задач, с которыми способны справиться. Эти методы обсуждаются в главе 10.

1.7.3. Генеративно-состязательные сети

Генеративно-состязательные сети (generative adversarial network, GAN) — это модель порождения данных, фактически построенная на основе двух моделей, состязающихся между собой в своеобразной игре. Этими двумя "игроками" являются генератор и дискриминатор. Генератор принимает гауссовский шум в качестве входа и выдает на выходе *сгенерированный* образец, похожий на пример из базовой обучающей выборки данных. В роли дискриминатора обычно выступает какой-нибудь вероятностный классификатор, например на основе логистической регрессии, который должен научиться отличать реальные образцы из базового набора данных от сгенерированных. Генератор пытается генерировать как можно более реальные образцы. Его задача состоит в том, чтобы обмануть дискриминатор, тогда как задача дискриминатора — идентифицировать поддельные образцы, несмотря на все попытки генератора обмануть его. Суть процесса можно понять, рассматривая его как состязательную игру между генератором и дискриминатором, тогда как ее формальной оптимизационной моделью является обучение задаче минимакса. Получение конечной обученной модели обеспечивается достижением *равновесия Нэша* в этой минимаксной игре. В типичных случаях точкой равновесия является та, в которой дискриминатору не удается отличать поддельные образцы от реальных.

Подобные методы позволяют создавать реалистично выглядящие фантазийные образцы на основе базового набора данных и обычно используются в задачах обработки изображений. Например, если использовать для тренировки набор данных, содержащий изображения спален, то сеть породит реалистичные изображения спален, которые на самом деле не являются частью базового набора. Поэтому такой подход вполне подходит для использования в художественных или творческих целях. Также возможна тренировка этих методов на специфических типах контекста, таких как метки, текстовые подписи или изображения с отсутствующими деталями. Во всех случаях используются пары тренировочных объектов. Типичным примером такой пары может служить подпись к рисунку

(контекст) и изображение (базовый объект). Точно так же подобными парами могут быть эскизы объектов и их фактические фотографии. Поэтому, начав с набора данных, который содержит изображения различных типов животных, снабженные подписями, можно создать фантазийное изображение, не являющееся частью базового набора данных, по контекстной подписи “синяя птица с острыми когтями”. Или же, начав со сделанного художником наброска дамской сумочки, можно получить ее реалистичное цветное изображение. Генеративно-состязательные сети обсуждаются в главе 10.

1.8. Два показательных эталонных теста

Среди обсуждаемых в литературе по нейронным сетям результатов эталонных тестов доминируют данные, относящиеся к области компьютерного зрения. Нейронные сети могут тестироваться на наборах данных традиционного машинного обучения, таких как репозиторий UCI [601], однако наблюдается тенденция использовать для этой цели преимущественно хорошо визуализируемые наборы данных. Несмотря на наличие широкого разнообразия наборов данных в виде текста и изображений, два из них занимают особое положение, поскольку ссылки на них повсеместно встречаются в статьях по глубокому обучению. И хотя оба этих набора связаны с компьютерным зрением, первый из них достаточно прост, чтобы его можно было использовать для тестирования обычных приложений, не имеющих отношения к зрению.

1.8.1. База данных рукописных цифр MNIST

База данных *MNIST* (от *Modified National Institute of Standards and Technology*) — это обширная база данных рукописных цифр [281]. Как следует из самого названия, набор данных был создан путем изменения оригинальной базы рукописных цифр, созданной Национальным институтом стандартов и технологий США (NIST). Набор содержит 60 000 тренировочных и 10 000 тестовых изображений. Каждое изображение получено сканированием рукописных цифр от 0 до 9, различия между которыми являются результатом того, что цифры были написаны различными людьми. Для этого были привлечены сотрудники Бюро переписи населения США и студенты американских вузов. Исходные черно-белые изображения, полученные от NIST, были нормализованы таким образом, чтобы они умещались в пределах окна размером 20×20 пикселей с соблюдением первоначального форматного соотношения, а затем центрированы в изображение размером 28×28 пикселей путем вычисления центра масс пикселей и его переноса в центр поля размером 28×28. Каждый из этих 28×28 пикселей имеет значение от 0 до 255, в зависимости от того, какую позицию на серой шкале он занимает. С каждым значением связана метка, соответствующая одной из десяти цифр.

Примеры цифр, содержащихся в базе данных MNIST, приведены на рис. 1.20. Это довольно небольшой набор данных, содержащий только простые объекты в виде цифр. Поэтому кое-кто мог бы назвать базу данных MNIST "игрушечной". Однако небольшой размер и простота набора одновременно являются его преимуществом, поскольку его можно легко использовать в качестве лаборатории для быстрого тестирования алгоритмов машинного обучения. Кроме того, дополнительная простота этого набора, обусловленная (приблизительным) центрированием цифр, облегчает тестирование алгоритмов, не связанных с компьютерным зрением. Алгоритмы компьютерного зрения требуют использования специальных предположений, таких как предположение о трансляционной инвариантности. Простота данного набора делает подобные предположения излишними. Как метко заметил Джефф Хинтон [600], исследователи в области нейронных сетей используют базу данных MNIST во многом подобно тому, как биологи используют дрозофилу для быстрого получения предварительных результатов (прежде чем выполнять серьезные тесты на более сложных организмах).

Рис. 1.20. Примеры рукописных цифр, хранящихся в базе данных MNIST

Несмотря на то что матричное представление каждого изображения подходит для сверточной нейронной сети, его также можно преобразовать в многомерное представление, имеющее $28 \times 28 = 784$ измерений. При таком преобразовании часть пространственной информации теряется, но эта потеря не сказывается значительно на результатах (по крайней мере, в случае набора данных MNIST ввиду его относительной простоты). И действительно, применение простого метода опорных векторов к 784-мерному представлению демонстрирует впечатляющий результат с ошибкой всего лишь 0,56%. Применение простой двухслойной нейронной сети для обработки многомерного представления

(без использования пространственной структуры изображения) обычно приводит к худшим результатам по сравнению с методом опорных векторов в широком диапазоне выбора значений параметров! Глубокая нейронная сеть без какой-либо сверточной архитектуры обеспечивает снижение ошибки до уровня 0,35% [72]. Более глубокие нейронные сети и сверточные нейронные сети (учитывающие пространственную структуру изображений) могут снизить ошибку до уровня 0,21% за счет использования ансамбля из пяти сверточных сетей [402]. Таким образом, даже в случае столь простого набора данных видно, что относительная производительность нейронных сетей по сравнению с традиционными методами машинного обучения чувствительна к специфике используемой архитектуры.

Наконец, следует отметить, что 784-мерное представление без учета пространственной структуры изображений набора MNIST используется для тестирования всех типов алгоритмов нейронных сетей, помимо тех, которые применяются в области компьютерного зрения. И хотя 784-мерное (плоское) представление не подходит для задач, связанных со зрением, его можно использовать для тестирования общей эффективности универсальных (т.е. не ориентированных на задачи, связанные со зрением) алгоритмов нейронных сетей. Например, базу данных MNIST часто используют для тестирования типичных, а не только сверточных автокодировщиков. Даже если для реконструкции изображения с помощью автокодировщика используется его непространственное представление, результаты по-прежнему можно визуализировать, используя информацию об исходных позициях реконструируемых пикселей, тем самым создавая впечатление, будто алгоритм работает непосредственно с данными. Такое визуальное исследование часто помогает ученым пролить свет на особенности, которые не удастся заметить при работе с произвольными наборами данных наподобие тех, которые получают из репозитория UCI Machine Learning Repository [601]. В этом смысле набор данных MNIST имеет более широкую применимость, чем многие другие типы наборов данных.

1.8.2. База данных ImageNet

ImageNet [581] — это огромная база данных, которая содержит свыше 14 миллионов изображений, образующих примерно 1000 различных категорий. Ее покрытие классов настолько широко, что она охватывает большинство типов изображений того, что может встретиться в повседневной жизни. Эта база данных организована в соответствии с иерархией существительных, используемой в электронном тезаурусе *WordNet* [329] — базе данных, содержащей соотношения между словами английского языка. Ее основной словарной единицей является не слово, а так называемый синонимический ряд — *синсет* (от англ. *synonym set*). Иерархия WordNet успешно использовалась для решения

задач машинного обучения в области обработки естественного языка, и поэтому построение набора изображений на основе используемых в ней соотношений между словами представляется вполне естественным.

Проект ImageNet известен проводимыми под его эгидой ежегодными соревнованиями *ImageNet Large ScaleVisual Recognition Challenge* (ILSVRC) [582]. Эти соревнования высоко ценятся сообществом специалистов по обработке изображений, и в них принимают участие большинство групп, занимающихся исследованиями в области компьютерного зрения. На этих соревнованиях были впервые представлены многие из современных передовых архитектур, ориентированных на распознавание образов, в том числе методы, продемонстрировавшие превосходство над человеком при решении некоторых узких задач, таких как классификация объектов [184]. Ввиду широкой доступности известных результатов, полученных с использованием этого набора данных, сравнение с ними является популярной альтернативой эталонному тестированию. Некоторые из современных алгоритмов, которые были представлены на соревнованиях ImageNet, будут рассмотрены в главе 8 при обсуждении сверточных нейронных сетей.

Другим немаловажным фактором, обусловливающим ценность набора данных ImageNet, является то, что ввиду своего большого размера и широкого разнообразия содержащихся в нем изображений он достаточно полно представляет все наиболее важные визуальные концепции в области обработки изображений. Как следствие, его часто используют для тренировки сверточных нейронных сетей, после чего предобученную сеть можно использовать для извлечения признаков из любого изображения. Это представление изображения определяется скрытыми активациями в предпоследнем слое нейронной сети. Такой подход позволяет создавать новые многомерные представления наборов данных изображений, пригодных для использования традиционными методами машинного обучения. Его можно рассматривать как своего рода переносимое обучение, где визуальные концепции, скрытые в наборе данных ImageNet, переносятся на неизвестные объекты, с которыми работают другие приложения.

1.9. Резюме

Несмотря на то что искусственную нейронную сеть можно рассматривать как имитацию процесса обучения в живых организмах, для более непосредственного понимания принципов работы этих сетей лучше представлять их в виде вычислительных графов. Такие вычислительные графы осуществляют рекурсивную композицию простых функций для обучения более сложным функциям. Поскольку вычислительные графы параметризованы, обычно задача сводится к обучению параметров графа посредством оптимизации функции потерь. Простейшими типами нейронных сетей часто являются базовые модели

машинного обучения, такие как регрессия по методу наименьших квадратов. Реальная мощь нейронных сетей проявляется в полной мере при использовании более сложных комбинаций базовых функций. Обучение параметров таких сетей ведется с использованием алгоритма динамического программирования, известного как метод обратного распространения ошибки. С обучением моделей нейронных сетей связаны определенные трудности, такие как переобучение и нестабильность процесса тренировки. Достижения последних лет в области разработки алгоритмов позволили несколько снизить остроту этих проблем. Проектирование методов глубокого обучения в таких специфических областях, как обработка текста и изображений, требует использования тщательно продуманных архитектур. Примерами таких архитектур могут служить рекуррентные и сверточные нейронные сети. В случае задач с динамическими условиями, требующими обучения принятию последовательности решений, могут быть полезными такие методы, как обучение с подкреплением.

1.10. Библиографическая справка

Понимание принципов работы нейронных сетей требует хорошего знания алгоритмов машинного обучения, особенно линейных моделей на основе градиентного спуска. Элементарное изложение методов машинного обучения содержится в [2, 3, 40, 177]. Многочисленные обзоры и сведения общего характера относительно использования нейронных сетей в различных контекстах приведены в [27, 28, 198, 277, 345, 431]. Классическими книгами по нейронным сетям для распознавания образов являются [41, 182], тогда как изложение более современных взглядов на перспективы глубокого обучения можно найти в [147]. В вышедшей недавно книге по интеллектуальному анализу текста [6] также обсуждаются последние достижения в области глубокого обучения. Обзоры взаимосвязи глубокого обучения и вычислительной нейробиологии содержатся в [176, 239].

Алгоритм перцептрона был предложен Розенблаттом [405]. Для устранения проблем нестабильности были предложены карманный алгоритм [128], алгоритм Maxover [523] и другие методы на основе понятия зазоров [123]. К числу других ранних алгоритмов аналогичного характера относятся алгоритмы Уидроу — Хоффа [531] и Уинноу [245]. Алгоритм Уинноу использует мультипликативные обновления вместо аддитивных и особенно полезен в тех случаях, когда многие признаки являются несущественными. Первоначальная идея обратного распространения ошибки основывалась на идее дифференцирования композиции функций, разработанной в теории управления [54, 237]. Использование динамического программирования для градиентной оптимизации переменных, связанных между собой посредством направленного ациклического графа, стало стандартной практикой с 60-х годов прошлого столетия. Однако

возможностей использования этих методов для тренировки нейронных сетей в то время еще не существовало. В 1969 году Минский и Пейперт опубликовали книгу по перцептрону [330], в которой высказали крайне негативную точку зрения в отношении возможности надежной тренировки многослойных нейронных сетей. В книге было продемонстрировано, что выразительные возможности одиночного перцептрона ограничены, а о том, как тренировать несколько слоев перцептрона, никто ничего не знал. Минский был влиятельной фигурой в мире искусственного интеллекта, и отрицательный тон его книги внес свой вклад в наступление длительного застоя в этой области. Адаптация методов динамического программирования к методу обратного распространения ошибки в нейронных сетях была впервые предложена Полом Вербосом в его докторской диссертации в 1974 году [524]. Однако работе Вербоса не удалось преодолеть сильного предубеждения против нейронных сетей, которое к тому времени уже успело укорениться. Алгоритм обратного распространения ошибки был вновь предложен Румельхартом и др. в 1986 году [408, 409]. Работа Румельхарта примечательна изяществом своего представления, и в ней были даны ответы на некоторые из вопросов, поднятых Минским и Пейпертом. Это одна из причин того, почему статья Румельхарта рассматривается как значимое событие для метода обратного распространения ошибки, хотя она и не была первой, в которой этот метод был предложен. Обсуждение истории развития метода обратного распространения ошибки можно найти в книге Пола Вербоса [525].

К этому времени интерес к нейронным сетям возродился лишь частично, поскольку все еще оставались проблемы, связанные с их тренировкой. Тем не менее целые группы исследователей продолжали работы в этом направлении и еще до 2000-го года успели разработать большинство известных нейронных архитектур, включая сверточные нейронные сети, рекуррентные нейронные сети и сети LSTM. Эти методы все еще демонстрировали довольно умеренную точность ввиду ограниченности доступных данных и вычислительных возможностей. Кроме того, механизм обратного распространения ошибки оказался менее эффективным в отношении тренировки глубоких сетей по причине затухающих и взрывных градиентов. Однако к тому времени рядом авторитетных исследователей уже была высказана гипотеза о том, что с увеличением объемов доступных данных и вычислительных мощностей и связанного с этим сокращения длительности экспериментов с алгоритмами производительность существующих алгоритмов должна улучшиться. Появление фреймворков больших данных и мощных GPU в конце 2000-х годов положительно сказалось на исследованиях в области нейронных сетей. Уменьшение длительности экспериментов, обусловленное возросшими вычислительными мощностями, создало условия для применения таких уловок, как предварительное обучение моделей [198]. Очевидное для широкой публики воскрешение нейронных сетей произошло после

2011 года и ознаменовалось рядом ярких побед [255] на соревнованиях глубоких сетей по классификации изображений. Неизменные победы алгоритмов глубокого обучения на этих соревнованиях заложили фундамент для взрыва их популярности, свидетелями чего мы являемся в наши дни. Примечательно то, что отличия выигрышных архитектур от тех, которые были разработаны двумя десятилетиями ранее, можно охарактеризовать лишь как умеренные (но имевшие существенное значение).

Пол Вербос, пионер рекуррентных нейронных сетей, предложил оригинальную версию алгоритма обратного распространения ошибки во времени [526]. Основные идеи сверточных нейронных сетей были предложены в контексте *неокогнитрона* в [127]. Затем эта идея была обобщена до сети LeNet-5 — одной из первых сверточных нейронных сетей. Способность нейронных сетей выступать в качестве универсального аппроксиматора функций обсуждается в [208]. Благотворное влияние глубины на уменьшение количества параметров сети обсуждается в [340].

Теоретическая универсальность нейронных сетей была признана еще на ранних этапах их развития. Например, в одной из ранних работ было показано, что нейронная сеть с единственным скрытым слоем может быть использована для аппроксимации любой функции [208]. Дополнительный результат заключается в том, что некоторые нейронные архитектуры наподобие рекуррентных сетей обладают полнотой по Тьюрингу [444]. Последнее означает, что нейронные сети потенциально могут имитировать любой алгоритм. Разумеется, существуют многочисленные практические проблемы, связанные с тренировкой нейронной сети, объясняющие, почему столь впечатляющие теоретические результаты не переходят в реальную производительность. Самой главной из них является жадная к данным природа мелких архитектур, что проявляется в меньшей степени с увеличением глубины. Увеличение глубины можно рассматривать как своего рода регуляризацию, в которой нейронную сеть вынуждают идентифицировать повторяющиеся закономерности среди точек данных и обучаться им. Однако увеличение глубины нейронной сети затрудняет ее тренировку с точки зрения оптимизации. Обсуждение некоторых из этих проблем можно найти в [41, 140, 147]. Экспериментальная оценка, демонстрирующая преимущества более глубоких архитектур, предоставлена в [267].

1.10.1. Видеолекции

Существует большое количество бесплатных видеокурсов по теме глубокого обучения, доступных на таких сайтах, как YouTube и Coursera. Одним из двух наиболее авторитетных ресурсов является видеокурс Джеффа Хинтона на сайте Coursera [600]. Там вы найдете многочисленные предложения по глубокому обучению, а также рекомендации, касающиеся группы курсов, относящихся

к этой области. На момент выхода книги к этим предложениям был добавлен курс Эндрю Энга. Курс по сверточным нейронным сетям, разработанный в Стэнфордском университете, бесплатно доступен на сайте YouTube [236]. Стэнфордский курс, который ведут Карпаты, Джонсон и Фай-Фай [236], также посвящен нейронным сетям, однако его охват более широкой тематики нейронных сетей не может похвастаться достаточной полнотой. В начальных частях курса рассматриваются простейшие нейронные сети и методы тренировки.

Многие темы машинного обучения [89] и глубокого обучения [90] охватывают лекции Нандо Де Фрейтаса, доступные на сайте YouTube. Еще один интересный курс по нейронным сетям предложил Хьюго Ларошель из Шербрукского университета [262]. Также доступен курс, разработанный Али Ходзи из университета Ватерлоо [137]. Видеолекции Кристофера Маннинга по методам обработки естественного языка опубликованы на YouTube [312]. Кроме того, доступен курс Дейвида Силвера по обучению с подкреплением [619].

1.10.2. Программные ресурсы

Глубокое обучение поддерживают многочисленные программные фреймворки, такие как *Caffe* [571], *Torch* [572], *Theano* [573] и *TensorFlow* [574]. Также доступны расширения *Caffe* для Python и MATLAB. Библиотека Caffe, написанная на C++, была разработана в Калифорнийском университете в Беркли. Она реализует высокоуровневый интерфейс, с помощью которого можно задать архитектуру сети, и позволяет конструировать нейронные сети, используя небольшой по размеру код и относительно простые сценарии. Основной недостаток Caffe — ограниченный объем доступной документации. Библиотека Theano [35], препроцессор которой написан на Python, предоставляет в качестве интерфейсов такие высокоуровневые пакеты, как *Keras* [575] и *Lasagne* [576]. В ее основе лежит понятие вычислительных графов, и большинство предоставляемых ею возможностей основано на явном использовании этой абстракции. Библиотека TensorFlow [574], предложенная компанией Google, также в значительной степени ориентирована на вычислительные графы. Библиотека Torch [572] написана на высокоуровневом языке Lua и имеет относительно дружественный пользовательский интерфейс. В последние годы Torch несколько укрепила свои позиции по сравнению с остальными фреймворками. Поддержка GPU тесно интегрирована в Torch, что упрощает развертывание приложений на основе Torch на компьютерах с GPU. Многие из этих фреймворков содержат предварительно обученные модели компьютерного зрения и интеллектуального анализа данных, которые могут быть использованы для извлечения признаков. Многие готовые инструменты глубокого обучения доступны в репозитории *DeepLearning4j* [590]. Платформа *PowerAI* компании IBM предлагает фреймворки машинного и глубокого обучения, работающие поверх серверов

IBM Power Systems [599]. Следует отметить, что на момент выхода книги также предлагался бесплатный вариант этой платформы, доступный для использования в определенных целях.

1.11. Упражнения

1. Пусть имеется функция XOR, в которой две точки {(0, 0), (1, 1)} принадлежат к одному классу, а две другие точки {(1, 0), (0, 1)} — к другому. Покажите, как разделить два этих класса, используя функцию активации ReLU, аналогично тому, как это было сделано в примере на рис. 1.14.
2. Покажите, что для функции активации в виде сигмоиды и гиперболического тангенса (обозначенной как $\Phi(\cdot)$ в каждом из этих случаев) справедливы приведенные ниже равенства.

   А. Сигмоида: $\Phi(-v) = 1 - \Phi(v)$.

   Б. Гиперболический тангенс: $\Phi(-v) = -\Phi(v)$.

   В. Спрямленный гиперболический тангенс: $\Phi(-v) = -\Phi(v)$.
3. Покажите, что гиперболический тангенс — это сигмоида, масштабированная вдоль горизонтальной и вертикальной осей и смещенная вдоль вертикальной оси:

$$\tanh(v) = 2\text{sigmoid}(2v) - 1.$$

4. Пусть имеется набор данных, в котором две точки {(−1, −1), (1, 1)} принадлежат к одному классу, а две другие точки {(1, −1), (−1, 1)} — к другому. Начните со значений параметра перцептрона (0, 0) и выполните несколько обновлений по методу стохастического градиентного спуска с $\alpha = 1$. В процессе получения обновленных значений циклически обходите тренировочные точки в любом порядке.

   А. Сходится ли алгоритм в том смысле, что изменения значений целевой функции со временем становятся предельно малыми?

   Б. Объясните, почему наблюдается ситуация, описанная в п. *А*.
5. Определите для набора данных из упражнения 4, в котором двумя признаками являются (x_1, x_2), новое одномерное представление z, обозначаемое как

$$z = x_1 \cdot x_2.$$

   Является ли набор данных линейно разделимым в терминах одномерного представления, соответствующего z? Объясните важность нелинейных преобразований в задачах классификации.
6. Реализуйте перцептрон на привычном для вас языке программирования.

7. Покажите, что значение производной сигмоиды не превышает 0,25, независимо от значения аргумента. При каком значении аргумента сигмоида имеет максимальное значение?

8. Покажите, что значение производной функции активации *tanh* не превышает 1, независимо от значения аргумента. При каком значении аргумента функция активации *tanh* имеет максимальное значение?

9. Пусть имеется сеть с двумя входами, x_1 и x_2. Сеть имеет два слоя, каждый из которых содержит два элемента. Предположим, что веса в каждом слое устанавливаются таким образом, что в верхнем элементе каждого слоя для суммирования его входов применяется сигмоида, а в нижнем — функция активации *tanh*. Наконец, в единственном выходном узле для суммирования двух его входов применяется активация ReLU. Запишите выход этой нейронной сети в замкнутой форме в виде функции x_1 и x_2. Это упражнение должно дать вам представление о сложности функций, вычисляемых нейронными сетями.

10. Вычислите частную производную по x_1 функции в замкнутой форме, полученной в предыдущем упражнении. Практично ли вычислять производные для градиентного спуска в нейронных сетях, используя выражения в замкнутой форме (как в традиционном машинном обучении)?

11. Пусть имеется двухмерный набор данных, в котором все точки с $x_1 > x_2$ принадлежат к положительному классу, а все точки с $x_1 \leq x_2$ — к отрицательному. Истинным разделителем для этих двух классов является линейная гиперплоскость (прямая линия), определяемая уравнением $x_1 - x_2 = 0$. Создайте набор тренировочных данных с 20 точками, сгенерированными случайным образом в положительном квадранте единичного квадрата. Снабдите каждую точку меткой, указывающей на то, превышает или не превышает ее первая координата x_1 вторую координату x_2.

А. Реализуйте алгоритм перцептрона без регуляризации, обучите его на полученных выше 20 точках и протестируйте его точность на 1000 точках, случайно сгенерированных в единичном квадрате. Используйте для генерирования тестовых точек ту же процедуру, что и для тренировочных.

Б. Замените критерий перцептрона на кусочно-линейную функцию потерь в своей реализации тренировки и повторите определение точности вычислений на тех же тестовых точках, которые использовали перед этим. Регуляризация не используется.

В. В каком случае и почему вам удалось получить лучшую точность?

Г. Как вы считаете, в каком случае классификация тех же 1000 тестовых точек не изменится значительно, если использовать другой набор из 20 тренировочных точек?