Глава 2

Нейронные сети

§ 2.1. Введение

Пока что ни один, даже самый мощный и современный, компьютер не может по своим способностям сравниться с человеческим мозгом. Да, компьютеры гораздо быстрее выполняют арифметические операции, но если мы хотим научить компьютер чему-либо более сложному, более «естественному» — понимать естественный язык, узнавать людей, принимать решения в сложных ситуациях, обучаться в широком смысле этого слова — компьютер пасует перед человеком.

Почему так происходит? Однозначного ответа на этот вопрос, конечно, нет. Например, Роджер Пенроуз в книге «Новый ум короля» [1] рассматривает и отнюдь не отвергает возможность того, что человеческий мозг в своей «вычислительной» деятельности использует квантовые феномены; то есть каждый из нас носит в голове мощнейший из известных квантовых компьютеров.

Но даже если оставить квантовые вычисления за бортом, — всё-таки это вовсе не общепринятая теория — устройство нашего мозга предоставляет пищу для любопытных размышлений. Мозг состоит из нейронов, которые соединяются друг с другом в единую большую сеть. Когда нейрон возбуждается, он передаёт своим соседям электрические сигналы определённого вида. Те обрабатывают сигналы и передают дальше, и так далее.

Примечательно, что соотношение скоростей передачи сигнала и человеческой реакции таково, что на самом деле цепочка нейронов, которые успели бы возбудиться перед принятием решения, не может быть длиннее нескольких сот штук! То есть мозг добивается своей, простите за каламбур, головоломной сложности и эффективности не огромным количеством *последовательных* вычислений, как современные компьютеры, а удачными связями между своими нейронами, которые позволяют для решения каждой конкретной задачи задействовать цепочки небольшой глубины.

В истории науки нередки случаи, когда люди заимствовали инженерные решения у матушки-природы. Многие животные, сами того не подозревая, становились прообразами конструктивных элементов зданий, машин и прочей аппаратуры. Поэтому нет ничего удивительного, что люди решили использовать свои знания о структуре

мозга для того, чтобы имитировать его работу — а там, чем чёрт не шутит, может, компьютер и мыслить начнёт... Так и появились *искусственные нейронные сети*.[1]

§ 2.2. Перцептрон

Нейрон человеческого мозга возбуждается, если получает достаточно мощный электрический импульс. При этом он может быть соединён с несколькими другими нейронами, и импульс может получаться как суммарный импульс от входящих нейронов. Простейшая математическая модель одного нейрона — это *перцептрон* (perceptron).

Перцептрон реализует вышеописанную модель линейными функциями. Он подсчитывает некоторую линейную форму от своих входов и сравнивает её с заданным значением — *лимитом активации* (threshold). Если у перцептрона n входов, то в нём должны быть заданы n весов $w_1, w_2, \ldots, w_n$ и лимит активации w_0. Перцептрон выдаёт 1, если линейная форма от входов с коэффициентами w_i превышает $-w_0$ (минус нужен, чтобы перенести w_0 в левую часть уже со знаком «плюс»). Иначе говоря, выход перцептрона $o(x_1, \ldots, x_n)$ вычисляется так:

$$o(x_1, \ldots, x_n) = \begin{cases} 1, \text{ если } w_0 + w_1 x_1 + \ldots + w_n x_n > 0, \\ -1 \text{ в противном случае.} \end{cases}$$

В дальнейшем мы не будем различать w_i, $i = 1..n$, и w_0, а просто предположим, что у перцептрона есть ещё один вход x_0, на который всегда подаётся единица. Тогда условие срабатывания перцептрона можно будет записать как $\sum_0^n w_i x_i > 0$.

Пример 2.1. Примеры перцептронов.

Мы будем изображать перцептроны в виде узлов графа, помечая их значением w_0. Значения w_i мы будем указывать на входящих рёбрах.

На рис.2.1 изображён общий вид перцептрона, а также даны примеры перцептронов, которые реализуют дизъюнкцию и конъюнкцию. Для дизъюнкции достаточно сделать лимит активации ниже, чем любой из весов на входе; тогда единица, полученная на любом входе, приведёт к срабатыванию. А чтобы получилась конъюнкция n входов, нужно установить такой threshold, чтобы сумма $n-1$ входа была меньше него, а сумма всех n входов — уже больше. И дизъюнкция, и конъюнкция — частные случаи функций вида «m из n», которые легко реализуются перцептроном.

[1] Существует направление исследований, посвящённых более точной имитации работы мозга, чем то, о чём мы будем говорить в этой главе. Однако их цель — не создание эффективных алгоритмов машинного обучения, а исследование мыслительных процессов человека. Поэтому мы не будем касаться этой проблематики.

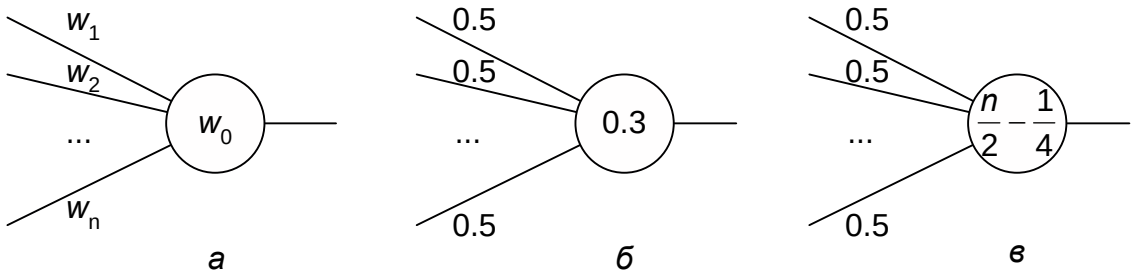

Рис. 2.1. Примеры перцептронов: *а* — общий вид перцептрона; *б* — перцептрон, реализующий дизъюнкцию; *в* — перцептрон, реализующий конъюнкцию.

Один перцептрон — это уже достаточно мощный инструмент: он может реализовать любую гиперплоскость, рассекающую пространство возможных решений. Если целевая функция (функция, которую мы пытаемся приблизить при помощи нейронной сети) позволяет *линейно отделить* свои положительные значения от отрицательных (т.е. если существует такая гиперплоскость, что все положительные значения лежат по одну её сторону, а отрицательные — по другую), то вполне достаточно будет создать сеть из *одного* перцептрона.

Мы объединяем перцептроны в сети, подавая выходы одних перцептронов на входы других. Эта конструкция оказывается очень мощной. Фактически, уже сеть глубины 2 — это всё, что может нам понадобиться для того, чтобы решить любую булевскую задачу.

Теорема 2.1. *Любая булевская функция представима в виде построенной из перцептронов искусственной нейронной сети глубины 2.*

Доказательство. См. упражнение 2. □

Главное ограничение перцептронов состоит в том, что они реализуют исключительно линейные функции. И комбинации линейных функций снова оказываются линейными. А искусственные нейронные сети могут использоваться и для куда более амбициозных задач. Но для этого им потребуются иные, более мощные элементы, о которых пойдёт речь ниже. А сейчас мы вспомним об основной стоящей перед нами задаче и рассмотрим *обучение* одного перцептрона.

§ 2.3. Обучение перцептрона

Уже понятно, что перцептроны отличаются друг от друга исключительно весами w_i. Значит, обучение должно заключаться в том, чтобы подправлять эти веса в соответствии с поведением перцептрона на тестовых примерах. При этом, если перцептрон

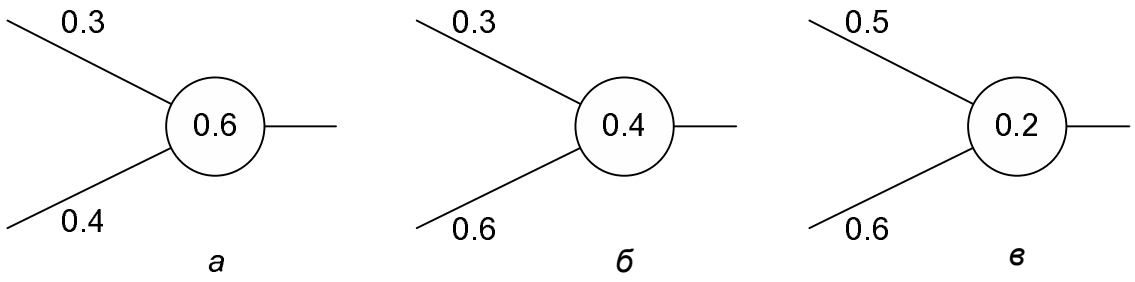

Рис. 2.2. Обучение перцептрона: *а* — перед началом обучения; *б* — после первого шага обучения; *в* — перцептрон, реализующий дизъюнкцию.

отработал правильно, веса, скорее всего, измениться не должны, а если неправильно, то должны сдвинуться в сторону желаемого.

Будем пока рассматривать сеть из одного перцептрона. Постановка задачи: научить такую сеть распознавать частично заданную булевскую функцию на основании данных о том, как справился перцептрон с тестовыми данными.

Простейший алгоритм так и называется в литературе — *правило обучения перцептрона* (perceptron training rule). На каждом шаге обучения вес w_i изменяется в соответствии с правилом

$$w_i \leftarrow w_i + \eta(t - o)x_i,$$

где t — значение целевой функции, o — выход перцептрона, $\eta > 0$ — небольшая константа (обычно 0.05–0.2), которая задаёт скорость обучения.

Пример 2.2. Обучение перцептрона.

Предположим, что мы хотим обучить перцептрон, изображённый на рис. 2.2а, реализовывать дизъюнкцию своих входов. Для этого нужно, чтобы значение и w_1, и w_2 стало больше значения лимита срабатывания w_0. Пусть $\eta = 0.1$, и у нас есть следующий набор тестовых данных:

$$\begin{aligned} x_1 = 0, \quad x_2 = 1 \quad &\Rightarrow \quad t = 1, \\ x_1 = 1, \quad x_2 = 0 \quad &\Rightarrow \quad t = 1. \end{aligned}$$

Первый же тест приведёт к ошибке: перцептрон выдаст $o = -1$, а искомый выход — $t = 1$. Таким образом, веса перцептрона будут подправлены следующим образом (см. рис. 2.2б):

$$\begin{aligned} w_0 \leftarrow w_0 + \eta(t - 0)x_0 = \quad & -0.6 + 0.1 \cdot (1 - (-1)) \cdot 1 = \quad -0.4, \\ w_1 \leftarrow w_1 + \eta(t - 0)x_1 = \quad & 0.3 + 0.1 \cdot (1 - (-1)) \cdot 0 = \quad 0.3, \\ w_2 \leftarrow w_2 + \eta(t - 0)x_2 = \quad & 0.4 + 0.1 \cdot (1 - (-1)) \cdot 1 = \quad 0.6. \end{aligned}$$

В полученном перцептроне один из весов на входе уже больше, чем лимит активации. Однако второй тест подаёт на этот вход ноль, и снова выяснится, что перцептрон не сработает.

`PerceptronTraining`$(\eta, \{x_i^j, t^j\}_{i=1,j=1}^{n,m})$

1. Инициализировать $\{w_i\}_{i=0}^{n}$ маленькими случайными значениями.
2. `WeightChanged` = `true`.
3. Пока `WeightChanged` = `true`:
   а) `WeightChanged` = `false`.
   б) Для всех j от 1 до m:
      (i) Вычислить
      $$o^j = \begin{cases} 1, \text{ если } w_0 + w_1 x_1^j + \ldots + w_n x_n^j > 0, \\ -1 \text{ в противном случае.} \end{cases}$$
      (ii) Если $o^j \neq t^j$:
         (A) `WeightChanged` = `true`.
         (B) Для каждого i от 0 до n изменить значение w_i по правилу
         $$w_i \leftarrow w_i + \eta(t^j - o^j)x_i^j.$$
4. Выдать значения $w_0, w_1, \ldots, w_n$.

Рис. 2.3. Алгоритм обучения перцептрона.

На этот раз веса придётся изменить так:

$$\begin{aligned} w_0 &\leftarrow w_0 + \eta(t-0)x_0 = -0.4 + 0.1 \cdot (1-(-1)) \cdot 1 = -0.2, \\ w_1 &\leftarrow w_1 + \eta(t-0)x_1 = 0.3 + 0.1 \cdot (1-(-1)) \cdot 1 = 0.5, \\ w_2 &\leftarrow w_2 + \eta(t-0)x_2 = 0.6 + 0.1 \cdot (1-(-1)) \cdot 0 = 0.6. \end{aligned}$$

Полученный перцептрон (рис. 2.2в) реализует дизъюнкцию своих входов и готов к дальнейшему использованию.

Может случиться так, что одной итерации по всем тестовым примерам будет недостаточно для того, чтобы исправить каждую ошибку (если бы в примере 2.2 η была бы равна не 0.1, а 0.01, веса были бы исправлены в нужную сторону, но недостаточно). Поэтому нужно запускать алгоритм по имеющимся тестовым примерам до тех пор, пока очередной прогон алгоритма по всем тестам не оставит все веса на месте. Одно применение алгоритма ко всем тестовым примерам называется *эпохой* (epoch). Получившийся алгоритм изображён на рис. 2.3. На вход ему подаётся набор из m тестовых примеров $\{x_i^j, t_j\}$ и скорость обучения η, а на выходе получаются значения w_i перцептрона, который линейно разделяет эти тестовые примеры.

В листинге 2.1 приведён код программы на языке Python, которая реализует обучение одного перцептрона. На вход подаются значение η и массив тестовых примеров в формате $[t^j, x_1^j, \ldots, x_n^j]$. В строке 4 значения весов инициализируются небольшими случайными значениями. В строке 9 мы читаем формат входных данных и заменяем для удобства t^j на единицу, то есть на x_0^j. Последняя строка — пример использования реализованной подпрограммы; на выходе должно получаться нечто вроде перцептрона, реализующего конъюнкцию трёх входов («нечто вроде», потому что функция задана частично, и вполне возможно, что при некоторых комбинациях начальных данных результат будет не идеальным).

Листинг 2.1. Обучение перцептрона на языке Python.

```
1   def PerceptronTraining(eta,x):
        import random
        w=[]
        for i in range(len(x[0])): w.append((random.randrange(-5,5))/50.0)
5       WeightsChanged=True
        while (WeightsChanged==True):
            WeightsChanged=False
            for xj in x:
                t,o,curx=xj[0],0,[1]+xj[1:len(xj)]
10              for i in xrange(len(w)): o+=w[i]*curx[i]
                if o>0: o=1
                else: o=-1
                if (o==t): continue
                WeightsChanged=True
15              for i in xrange(len(w)): w[i]+=eta*(t-o)*curx[i]
        return w
```

Алгоритм на рис. 2.3 сходится к правильному перцептрону всегда, когда это возможно.

Теорема 2.2. *Если конечное множество точек $C_1 \subset \{0,1\}^n$ можно в $\{0,1\}^n$ отделить гиперплоскостью от конечного множества точек $C_2 \subset \{0,1\}^n$, то алгоритм на рис. 2.3 за конечное количество шагов выдаёт параметры перцептрона, который успешно разделяет множества C_1 и C_2.*

Доказательство. Итак, мы предполагаем, что входы принадлежат к отделимым гиперплоскостью множествам; это означает, что существует такой вектор u (нормаль

к той самой гиперплоскости), что

$$\forall x \in C_1 \quad u^\top x > 0,$$
$$\forall x \in C_2 \quad u^\top x < 0.$$

Цель обучения перцептрона — сделать так, чтобы веса перцептрона w образовывали такой вектор u.

Прежде всего заменим C_2 на $-C_2 = \{-x \mid x \in C_2\}$. Это позволит нам объединить два неравенства в одно:

$$\forall x \in C \; w^\top x > 0,$$

где $C = C_1 \cup (-C_2)$.

Обозначим через $w^0, w^1, \ldots$ векторы весов перцептрона на соответствующих этапах обучения, через $x^0, x^1, \ldots$ — векторы тестовых примеров. Предположим (без потери общности), что $w^0 = 0$, все тестовые примеры принадлежат C, и на всех тестовых примерах $(w^k)^\top x^k < 0$, т.е. перцептрон не справляется с тестом (если он справляется с тестом, то веса перцептрона никак не меняются, поэтому такие тесты можно просто исключить из последовательности). В таком случае наше правило обучения выглядит как

$$w_i^{k+1} = w_i^k + \eta x_i^k,$$

и легко видеть, что в наших предположениях

$$w^k = \eta \sum_{j=0}^{k-1} x^j.$$

Идея доказательства состоит в том, чтобы получить две серии противоположных оценок на длину векторов $\|w^i\|$ и показать, что они не могут обе иметь место до бесконечности. Это и будет означать, что любая конечная последовательность тестов из двух фиксированных линейно отделимых множеств рано или поздно закончится, т.е. все последующие тесты перцептрон будет проходить успешно (напоминаем, что эти тесты исключены из нашего рассмотрения).[2]

Рассмотрим вектор u, являющийся решением, и обозначим

$$\alpha = \min_j u^\top x^j$$

(здесь мы используем тот факт, что разных тестов конечное число).

[2] Можно подумать, что «конечная последовательность тестов» закончится в любом случае; но напомним, что наш алгоритм повторяет тесты до тех пор, пока перцептрон не научится проходить их все. Речь идёт лишь о том, что в этой (возможно, бесконечной) последовательности тестов участвует конечное число различных векторов x^i.

Тогда

$$u^\top w^{k+1} = \eta u^\top \sum_{j=0}^{k} x^j \geq \eta \alpha k.$$

Применим неравенство Коши-Буняковского $\|x\|^2\|y\|^2 \geq (x \cdot y)^2$:

$$\|u\|^2\|w^{k+1}\|^2 \geq (\eta\alpha k)^2, \quad \|w^{k+1}\|^2 \geq \frac{(\eta\alpha k)^2}{\|u\|^2}.$$

Иными словами, на каждой итерации длина вектора w^k возрастает по квадратичному закону (пропорционально k^2). Остальные члены правой части неравенства — константы.

С другой стороны, $w^{k+1} = w^k + \eta x^k$. Поскольку $(w^k)^\top x^k < 0$,

$$\|w^{k+1}\|^2 = \|w^k\|^2 + 2\eta(w^k)^\top x^k + \eta^2\|x^k\|^2 \leq \|w^k\|^2 + \eta^2\|x^k\|^2.$$

Следовательно, $\|w^{k+1}\|^2 - \|w^k\|^2 \leq \eta^2\|x^k\|^2$. Суммируя по k, имеем:

$$\|w^{k+1}\|^2 \leq \eta^2 \sum_{j=0}^{k} \|x^k\|^2 \leq \eta^2\beta k,$$

где $\beta = \max_j \|x^j\|^2$ (мы опять воспользовались тем, что тестовых примеров конечное число).

Итак, у нас получились две оценки:

$$\frac{(\eta\alpha)^2}{\|u\|^2}k^2 \leq \|w^{k+1}\|^2 \leq \eta^2\beta k.$$

Очевидно, что рано или поздно с ростом k (k — это натуральное число) эти оценки войдут в противоречие друг с другом. Это значит, что последовательность применения одних и тех же тестовых примеров не может быть бесконечной: рано или поздно все примеры, если это было вообще возможно, будут перцептроном проходиться правильно.

□