

# Оглавление

<b>1</b>	<b>Введение</b>	<b>3</b>
1.1	Машинное обучение . . . . .	3
1.2	Виды задач . . . . .	4
1.3	Выбор модели, переобучение . . . . .	4
<b>2</b>	<b>Классическое обучение с учителем</b>	<b>5</b>
2.1	Линейные модели . . . . .	5
2.1.1	Почему модели линейные? . . . . .	5
2.1.2	Линейная регрессия и МНК . . . . .	5
2.1.3	точный аналитический метод . . . . .	6
2.1.4	приближенный численный метод . . . . .	6
2.1.5	Стохастический градиентный спуск . . . . .	7
2.1.6	Регуляризация . . . . .	9
2.1.7	Разрежение весов в $L^1$ -регуляризации . . . . .	10
2.1.8	Другие лоссы . . . . .	10
2.1.9	Линейная классификация . . . . .	10
2.1.10	Логистическая регрессия . . . . .	12

2.1.11	Многоклассовая классификация . . . . .	13
2.1.12	Многоклассовая логистическая регрессия . . . . .	14
<b>3</b>	<b>Вероятностные модели</b>	<b>15</b>
3.1	Первое знакомство с вероятностными моделями . . . . .	15
3.1.1	Случайность как источник несовершенства модели . . . . .	15
3.1.2	Условное распределение на таргет, непрерывный случай . . . . .	15
3.1.3	Более сложные вероятностные модели . . . . .	16
3.1.4	Оценка максимального правдоподобия = оптимизация функции потерь . . . . .	17
<b>4</b>	<b>Введение в глубинное обучение</b>	<b>18</b>
4.1	Первое знакомство с полносвязными нейросетями . . . . .	18
4.1.1	Основные определения . . . . .	18
<b>5</b>	<b>Основные алгоритмы</b>	<b>20</b>
5.1	Backpropagation . . . . .	20
5.1.1	Вступление . . . . .	20
5.1.2	Почему вообще нужно задумываться о том, как работает ВР? . . . . .	20
5.1.3	Простейший пример . . . . .	20
5.1.4	Слой $L - 1$ . . . . .	21
5.1.5	Слой $L - 2$ . . . . .	26

# Глава 1

## Введение

### 1.1 Машинное обучение

**Машинное обучение** – это наука, изучающая алгоритмы, автоматически улучшающиеся благодаря опыту.

Большинство решений задач можно представить в виде функции:

$$\text{Примеры (samples)} \rightarrow \text{Предсказания (targets)}.$$

Данная функция – **модель**, а набор примеров – **обучающая выборка (dataset)**.

Обучающая выборка = Объекты + Ответы.

Качество таких предсказаний измеряют **метриками** – функциями, которые показывают насколько полученные предсказания похожи на правильные ответы. Примером метрики является **среднее абсолютное отклонение**:

$$MAE(f, X, y) = L(f, X, y) = \frac{1}{N} \sum_{i=1}^N |f(x_i) - y_i|.$$

На сегодня достаточно знать два типа моделей – **градиентный бустинг на решающих деревьях** и **нейросетевые модели**.

Метрику, которую используют при поиске оптимальной модели – **функция потерь, лосс-функцией (loss)**.

## 1.2 Виды задач

Определенные выше задачи – **обучение с учителем (supervised learning)**, так как правильные ответы были даны заранее. Виды таких обучений:

- $\mathbb{Y} = \mathbb{R}$  или  $\mathbb{Y} = \mathbb{R}^M$  – **регрессия**;
- $\mathbb{Y} = \{0, 1\}$  – **бинарная классификация**;
- $\mathbb{Y} = \{1, \dots, K\}$  – **многоклассовая (multiclass) классификация**;
- $\mathbb{Y} = \{0, 1\}^K$  – **многоклассовая классификация с пересекающимися классами (multilabel classification)**;
- $\mathbb{Y}$  – конечное упорядоченное множество – **ранжирование**.

Имеется другой класс задач – **обучение без учителя (unsupervised learning)** – для которой известны только данные, а ответы отсутствуют. Одним из примеров является *кластеризация* – задача разделения объектов на группы, обладающие некоторыми свойствами.

## 1.3 Выбор модели, переобучение

**Обобщающаяся способность** модели – способность модели учить общие закономерности и давать адекватные предсказания. Выборку для этого делят на две части: **обучающая выборка** и **тестовая выборка (train и test)**.

Такой подход позволяет отделить модели, которые просто удачно подстроились к обучающим данным, от моделей, в которых произошла **генерализация (generalization)**, то есть от таких, которые на самом деле кое-что поняли о том, как устроены данные, и могут выдавать полезные предсказания для объектов, которые не видели.

**Переобученный алгоритм** – алгоритм, избыточно подстроившийся под данные.

## Глава 2

# Классическое обучение с учителем

### 2.1 Линейные модели

#### 2.1.1 Почему модели линейные?

Семейства линейных функций:

$$y = \omega_1 \cdot x_1 + \dots + \omega_D \cdot x_D + \omega_0$$

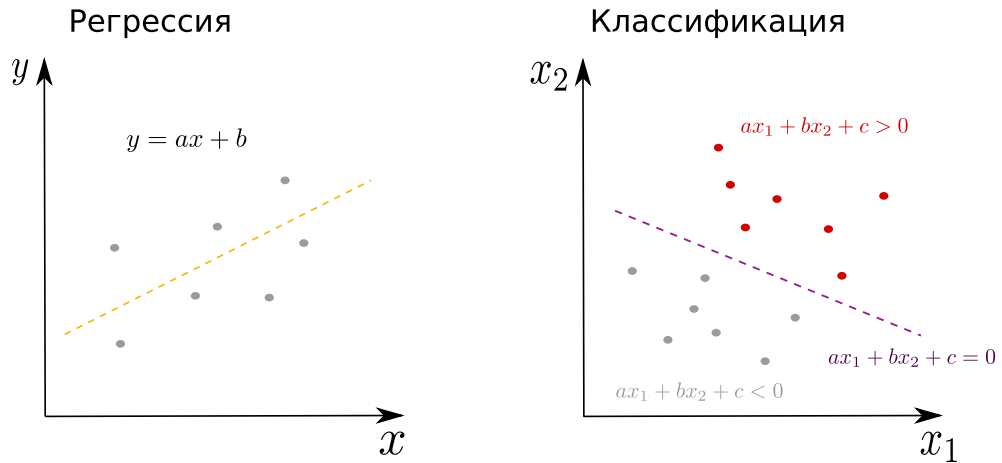
где  $y$  – целевая переменная,  $(x_1, \dots, x_D)$  – **вектор признаков**,  $\omega_0, \dots, \omega_D$  – параметры модели.  $\omega = (\omega_1, \dots, \omega_D)$  – **вектор весов**,  $\omega_0$  – **сдвиг (bias)**. Другая запись функции:

$$y = \langle x, \omega \rangle + \omega_0.$$

#### 2.1.2 Линейная регрессия и МНК

Пусть  $y = (y_i)_{i=1}^N \in \mathbb{R}^N$ ,  $X = (x_i)_{i=1}^N \in \mathbb{R}^{N \times D}$  – матрица объекты-признаки (i-ая строка – вектор признаков i-го объекта). Наша линейная функция выглядит следующим образом:

$$f_\omega(x_i) = \langle \omega, x_i \rangle + \omega_0.$$



Сделаем так, чтобы наша функция как можно ближе приближала нашу зависимость. Научимся измерять «плохость» модели. **Функция потерь (loss function)** – функция «плохости» модели.

В качестве лосса возьмем:

$$L(f, X, y) = \|y - f_\omega(X)\|_2^2 = \|y - f_\omega(X)\|_2^2 = \sum_{i=1}^N (y_i - \langle x_i, \omega \rangle)^2$$

Минимизируем нашу функцию потерь:  $\|y - f_\omega(X)\|_2^2 \rightarrow \min_\omega$ .

### 2.1.3 точный аналитический метод

Приравняем к нулю градиент по  $\omega$  и получим формулу для оптимального вектора весов:

$$\omega_* = (X^T X)^{-1} X^T y.$$

Действие получение точного решения вычислительно трудное, а также возможно обратная матрица плохо обусловлена (погрешность вычисления вектора весов зависит от квадрата числа обусловленности).

### 2.1.4 приближенный численный метод

Используем то, что градиент функции в точке направлен в сторону её роста. То есть, мы можем улучшить приближенное оптимальное значение параметра  $\omega$ ,

немного сдвинув его в направлении антиградиента:

$$\omega_j \mapsto \omega_j - \alpha \frac{d}{d\omega_j} L,$$

где  $\alpha$  – **темп обучения**. Описанный алгоритм называется **градиентным спуском**.

Градиент функции потерь:

$$\nabla_w L = 2X^T (X\omega - y).$$

**Алгоритм градиентного спуска:**

```
w = random_normal()
repeat S times:
    f = X.dot(w)
    err = f - y
    grad = 2 * X.T.dot(err) / n
    w -= alpha * grad
```

Вычислительная сложность градиентного спуска –  $O(NDS)$ , где  $N$  – длина выборки,  $D$  – число признаков,  $S$  – число итераций.

Сложность по памяти –  $O(ND)$ .

### 2.1.5 Стохастический градиентный спуск

Возникает идея заменить градиент его оценкой на подвыборке (**batch, mini-batch**). Используют линейный проход по выборке, при котором на очередных примерах ( $B$  – размер батча) вычисляется градиент и производится обновление весов модели. Количество шагов задают количеством **эпох**  $E$  – полным проходом по выборке.

**Алгоритм:**

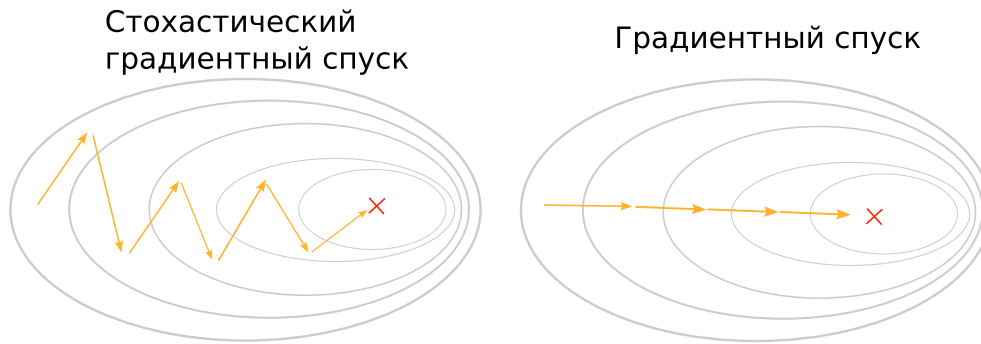
```
w = normal(0,1)
repeat E times:
    for i = B, i <= n, i += B:
        X_batch = x[i-B : i]
        Y_batch = y[i-B : i]
        f = X_batch.dot(w)
        err = f - Y_batch
        grad = 2 * X_batch.T.dot(err) / n
```

`w -= alpha * grad`

Сложность по времени –  $O(NDE)$ .

Сложность по памяти –  $O(BD)$  (всю выборку не надо держать в памяти, достаточно только загружать лишь текущий батч)





### 2.1.6 Регуляризация

Бывает так, что признаки могут быть приближенно линейно зависимыми – **мультиколлинеарность**. Симметричная матрица, которую впоследствии должны обратить, близка к вырожденной. Некоторые из собственных чисел будут близки к нулю. Из этого будет следовать умножение на  $\frac{1}{\lambda_i}$  в нахождении вектора весов, что влечет к появлению больших по модулю чисел.

Чтобы справиться с этой задачей **регуляризуют** – добавляют дополнительное ограничение на величину вектора весов. Вместо исходной задачи имеем:

$$\min_{\omega} L(f, X, y) = \min_{\omega} (\|X\omega - y\|_2^2 + \lambda \|\omega\|_k^k),$$

где  $\lambda \|\omega\|_k^k$  – **регуляризационный член**,  $\lambda$  – **коэффициент регуляризации**.

Отметим, что  $\omega_0$  не имеет смысла регуляризовать, т. е. например:

$$\|\omega\|_2^2 = \sum_{j=1}^D \omega_j^2$$

В случае  $L^2$ -регуляризации, продифференцировав новый лосс, получим «точное» решение:

$$\omega = (X^T X + \lambda I)^{-1} X^T y.$$

Градиент функции потерь:

$$\nabla_{\omega} L(f_{\omega}, X, y) = 2X^T(X\omega - y) + 2\lambda\omega$$

### 2.1.7 Разрежение весов в $L^1$ -регуляризации

Существует полезная особенность данной регуляризации: её применение приводит к тому, что у признаков, которые не оказывают большого влияния на ответ, вес в результате оптимизации получается равным нулю.

### 2.1.8 Другие лоссы

**Mean absolute error**, абсолютная ошибка:

$$MAE(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i|.$$

**Mean absolute percentage error**, относительная ошибка:

$$MAPE(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^N \left| \frac{\hat{y}_i - y_i}{y_i} \right|.$$

### 2.1.9 Линейная классификация

Рассмотрим классификацию на 2 класса. Обучим линейную модель строить плоскость, которая будет как можно лучше отделять объекты одного класса от другого.

Выборка, для которой возможна плоскость, которая идеально разделит классы, называется **линейно разделимой**.

Итоговое предсказание вычисляется по формуле:

$$y = \text{sign}\langle \omega, x_i \rangle.$$

Минимизируем число ошибок предсказателя:

$$\sum_i \mathbb{I}[y_i \neq \text{sign}\langle \omega, x_i \rangle] \rightarrow \min_{\omega}$$

Домножим обе части на  $y_i$  и упростим:

$$\sum_i \mathbb{I}[y_i \langle \omega, x_i \rangle < 0] \rightarrow \min_{\omega},$$

где  $M = y_i \langle \omega, x_i \rangle$  – **отступ (margin)** классификатора, ошибка – **misclassification loss**. То есть, имеем функцию:

$$F(M) = \mathbb{I}[M < 0] = \begin{cases} 1, & M < 0 \\ 0, & M \geq 0 \end{cases}$$

Мажорируем её более гладкой функцией (кусочно-постоянную функцию невозможно оптимизировать градиентными методами). Некоторые из них:

- Перцептрон:  $\max(0, -M_i)$
- Hinge (SVM):  $\max(0, 1 - M_i)$

Лосс перцептрона с  $L^2$ -регуляризации:

$$L(\omega, x, y) = \lambda \|\omega\|_2^2 + \sum_i \max(0, -y_i \langle \omega, x_i \rangle)$$

Градиент функции потерь:

$$\nabla_{\omega} L(\omega, x, y) = 2\lambda\omega + \sum_i \begin{cases} 0, & y_i \langle \omega, x_i \rangle > 0 \\ -y_i x_i, & y_i \langle \omega, x_i \rangle \leq 0 \end{cases}$$

Возникает желание не только найти разделяющую прямую, но и постараться провести её на одинаковом удалении от обоих классов, то есть максимизировать минимальный отступ. Для этого слегка поменяем функцию ошибки на Hinge. Лосс в этом случае:

$$L(\omega, x, y) = \lambda \|\omega\|_2^2 + \sum_i \max(0, 1 - y_i \langle \omega, x_i \rangle)$$

Градиент:

$$\nabla_{\omega} L(\omega, x, y) = 2\lambda\omega + \sum_i \begin{cases} 0, & 1 - y_i \langle \omega, x_i \rangle \leq 0 \\ -y_i x_i, & 1 - y_i \langle \omega, x_i \rangle > 0 \end{cases}$$

Ближайшие к плоскости правильно классифицированные объекты называются **опорные векторы (support vectors)**. Итоговое положение задается такими обучающимися примерами – такой зовется **метод опорных векторов (support vector machine)**.

### 2.1.10 Логистическая регрессия

Появляется желание посмотреть на классификацию, как на задачу предсказания вероятностей. Научим линейную модель предсказывать какой-то объект, связанный с вероятностью, но с диапазоном значений  $(-\infty, +\infty)$ . Таким объектом является **logit** или **log odds**:  $\log\left(\frac{p}{1-p}\right)$ . Тогда:

$$\langle \omega, x_i \rangle = \log\left(\frac{p}{1-p}\right)$$

$$e^{\langle \omega, x_i \rangle} = \frac{p}{1-p}$$

$$p = \frac{1}{1 + e^{-\langle \omega, x_i \rangle}}.$$

Данная функция называется **сигмоидой** и обозначается  $\sigma(z) = \frac{1}{1+e^{-z}}$ .

Для наилучшей оптимизации  $\omega$  применим метод максимума правдоподобия для распределения Бернулли. Оно позволяет нам понять, насколько вероятно получить данные значения таргета  $y$  при данных  $X$  и весах  $\omega$ :

$$p(y|X, \omega) = \prod_i p(y_i|x_i, \omega).$$

Для распределения Бернулли:

$$p(y|X, \omega) = \prod_i p_i^{y_i} (1 - p_i)^{1-y_i},$$

где  $p_i$  – вероятность, посчитанная из ответов модели. Перейдём к логарифмическому правдоподобию и подставим прошлую формулу:

$$\begin{aligned} l(\omega, X, y) &= \sum_i (y_i \log(p_i) + (1 - y_i) \log(1 - p_i)) = \\ &= \sum_i (y_i \log(\sigma(\langle \omega, x_i \rangle)) + (1 - y_i) \log(1 - \sigma(\langle \omega, x_i \rangle))) \end{aligned}$$

Так как  $\sigma(-z) = \frac{1}{1+e^z} = \frac{e^{-z}}{e^{-z}+1} = 1 - \sigma(z)$ , то:

$$l(\omega, X, y) = \sum_i (y_i \log(\sigma(\langle \omega, x_i \rangle)) + (1 - y_i) \log(\sigma(-\langle \omega, x_i \rangle))).$$

В максимуме данной отрицательной функции находится наше оптимизиран-

ное значение  $\omega$ . Для минимизации функция лосса выглядит следующим образом:

$$L(\omega, X, y) = - \sum_i (y_i \log(\sigma(\langle \omega, x_i \rangle)) + (1 - y_i) \log(\sigma(-\langle \omega, x_i \rangle))).$$

Градиент функции:

$$\nabla_{\omega} L(y, X, \omega) = - \sum_i x_i (y_i - \sigma(\langle \omega, x_i \rangle)).$$

Наше предсказание лежит в пределе от 0 до 1. Находить порог между этими двумя классами лучше отдельно, для уже построенной регрессии, минимизируя нужную метрику на отложенной тестовой выборке.

### 2.1.11 Многоклассовая классификация

Каждый объект выборки относится к одному из  $K$  классов:  $\mathbb{Y} = 1, \dots, K$ . Сведем многоклассовую задачу к набору бинарных. Существует два способа это сделать: **one-vs-all** и **all-vs-all**.

Рассмотрим первый способ. Обучим  $K$  линейных классификаторов  $b_1(x), \dots, b_K(x)$ , выдающих оценки принадлежности классам  $1, \dots, K$  соответственно. Они имеют вид:

$$b_k(x) = \text{sgn}(\langle \omega_k, x \rangle + \omega_{0k}).$$

Классификатор с номером  $k$  будет обучать по выборке  $(x_i, 2\mathbb{I}[y_i = k] - 1)$ .

Итоговый классификат выдает класс, соответствующий самому «уверенному» из алгоритмов:

$$a(x) = \arg \max_k (\langle \omega_k, x \rangle + \omega_{0k}).$$

Проблема данного подхода заключается в том, что каждый из классификаторов обучается на своей выборке, и их выходы имеют могут иметь разные масштабы, из-за чего сравнивать их будет неправильно.

Рассмотрим второй способ. Обучим  $C_K^2$  классификаторов  $a_{ij}(x)$ ,  $i, j = 1, \dots, K$ ,  $i \neq j$ . Данные классификаторы будем настраивать по подвыборке  $X_{ij} \subset X$ , содержащей только объекты классов  $i$  и  $j$ . Соответственно  $a_{ij}(x)$  будем выводить  $i$  или  $j$ . Чтобы классифицировать новый объект, подадим его на вход каждой из построенных бинарных классификаторов; в качестве ответа выберем тот класс,

за который наберется больше всего голосов:

$$a(x) = \arg \max_k \sum_{i=1}^K \sum_{j \neq i} \mathbb{I}[a_{ij}(x) = k].$$

### 2.1.12 Многоклассовая логистическая регрессия

Также построим  $K$  линейных классификаторов. Нашей целью является преобразование вектора оценок в вероятности. Можно сделать следующее:

$$\text{softmax}(z_1, \dots, z_K) = \left( \frac{\exp(z_1)}{\sum_{k=1}^K \exp(z_k)}, \dots, \frac{\exp(z_K)}{\sum_{k=1}^K \exp(z_k)} \right).$$

В этом случае вероятность  $k$ -го класса:

$$P(y = k|x, \omega) = \frac{\exp(\langle \omega_k, x \rangle + \omega_{0k})}{\sum_{j=1}^K \exp(\langle \omega_j, x \rangle + \omega_{0j})}.$$

Обучить эти веса предлагается с помощью метода максимального правдоподобия:

$$\sum_{i=1}^K \log P(y = y_i | x_i, \omega) \rightarrow \max_{\omega_1, \dots, \omega_K}.$$

## Глава 3

# Вероятностные модели

### 3.1 Первое знакомство с вероятностными моделями

#### 3.1.1 Случайность как источник несовершенства модели

Решаем задачу регрессии  $y \approx \langle x, \omega \rangle$ . Если мы не можем это точно измерить, то это случайный фактор. Тогда меняем приближенное на точное равенство следующим образом:

$$y = \langle x, \omega \rangle + \varepsilon,$$

где  $\varepsilon$  – некая случайная величина.

#### 3.1.2 Условное распределение на таргет, непрерывный случай

Пусть  $\varepsilon$  имеет плотность распределения  $\sim p_\varepsilon(t)$ .  $y$  превращается в случайную величину, зависящую от  $x$  и  $\omega$ , то есть можно говорить об условном распределении:

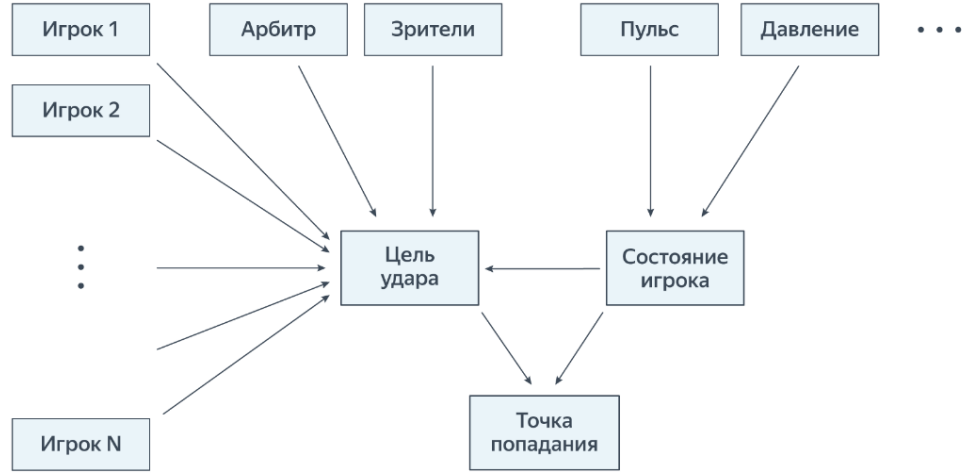
$$p_y(y|x, \omega).$$

Например, если имеем  $y = \langle x, \omega \rangle + \varepsilon$ ,  $\varepsilon \sim N(0, \sigma^2)$ , то  $y_i \sim N(\langle x_i, \omega \rangle, \sigma^2)$ . Это можно записать:

$$p(y_i|x_i, \omega) \sim N(y_i|\langle x_i, \omega \rangle, \sigma^2)$$

### 3.1.3 Более сложные вероятностные модели

В моделях функции могут принимать более сложнoзависимый характер – такое описывают с помощью графических моделей.



Распределив всё по определению условной вероятности:

$$\begin{aligned}
 p\left(\begin{matrix} \text{точка} \\ \text{попадания} \end{matrix}\right) &= \int p\left(\begin{matrix} \text{точка} & \text{цель} & \text{состояние} \\ \text{попадания} & \text{удара} & \text{игрока} \end{matrix}\right) d\left(\begin{matrix} \text{цель} \\ \text{удара} \end{matrix}\right) d\left(\begin{matrix} \text{состояние} \\ \text{игрока} \end{matrix}\right) = \\
 &= \int p\left(\begin{matrix} \text{точка} & \text{цель} & \text{состояние} \\ \text{попадания} & \text{удара} & \text{игрока} \end{matrix}\right) p\left(\begin{matrix} \text{цель} & \text{состояние} \\ \text{удара} & \text{игрока} \end{matrix}\right) d\left(\begin{matrix} \text{цель} \\ \text{удара} \end{matrix}\right) d\left(\begin{matrix} \text{состояние} \\ \text{игрока} \end{matrix}\right) = \\
 &= \int p\left(\begin{matrix} \text{точка} & \text{цель} & \text{состояние} \\ \text{попадания} & \text{удара} & \text{игрока} \end{matrix}\right) p\left(\begin{matrix} \text{цель} & \text{состояние} \dots \\ \text{удара} & \text{игрок 1, игрок 2} \\ \text{арбитр, зрители} \end{matrix}\right) p\left(\begin{matrix} \text{состояние} \\ \text{игрока} \end{matrix} \middle| \begin{matrix} \text{пульс} \\ \text{давление} \end{matrix}\right) \times \\
 &\times \left[ \prod_i p(\text{игрок } i) \right] p(\text{арбитр}) p(\text{зрители}) p(\text{пульс}) p(\text{давление}) \times \\
 &\times \left[ \prod_i d(\text{игрок } i) \right] d(\text{арбитр}) d(\text{зрители}) d(\text{пульс}) d(\text{давление})
 \end{aligned}$$



### 3.1.4 Оценка максимального правдоподобия = оптимизация функции потерь

Суть метода максимального правдоподобия (**maximum likelihood estimation**) состоит в том, чтобы найти такое  $\omega$ , для которого вероятность появления выборки  $y$  была бы максимальной:

$$\omega_{\text{MLE}} = \arg \max_{\omega} p(y|X, \omega).$$

Функция внутри правой части – **функция правдоподобия**. Если объекты функции правдоподобия независимы, то:

$$p(y|X, \omega) = p(y_1|x_1, \omega) \cdot \dots \cdot p(y_i|x_i, \omega).$$

Переходим к логарифму:

$$l(y|X, \omega) = \log(p(y_1|x_1, \omega)) + \dots + \log(p(y_i|x_i, \omega)) = \sum_{i=1}^N \log p_{\varepsilon}(y - f_{\omega}(x_i)).$$

Максимизация функции правдоподобия соответствует минимизации а это выражения можно проинтерпретировать, как функцию потерь.

## Глава 4

# Введение в глубинное обучение

### 4.1 Первое знакомство с полносвязными нейросетями

#### 4.1.1 Основные определения

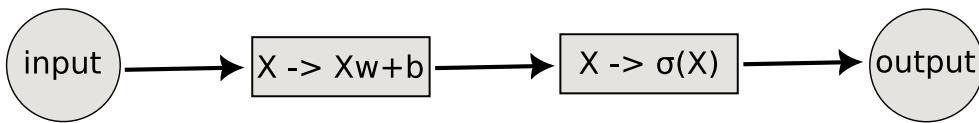
**Искусственная нейронная сеть** – сложная дифференцируемая функция, задающая отображение из исходного признакового пространства в пространство ответов, все параметры которой могут настраиваться одновременно и взаимосвязанно.

Нейронные сети обычно предстают в виде конструктора, состоящего из более-менее простых блоков (**слоёв, layers**). Их две простейшие разновидности:

- **Линейный слой (linear layer, dense layer)** – линейное преобразование над входящими данными:  $x \rightarrow xW + b$  ( $W \in \mathbb{R}^{d \cdot k}$ ,  $x \in \mathbb{R}^d$ ,  $b \in \mathbb{R}^k$ ).
- **Функция активации (activation function)** – нелинейное преобразование, поэлементно применяющееся к пришедшим на вход данным.

Таким образом совокупность таких простых блоков представляет собой **вычислительный граф (computational graph)**, где промежуточным вершинам соответствуют преобразования. Ниже дан пример вычислительного графа логисти-

ческой регрессии. Давайте разберемся, что тут происходит:



- Input – **вход** нейросети, который получает исходные данные.
- Далее применяются два слоя, которые превращают данные в **промежуточные представления (активации)**.
- Output – **выход** нейросети, который выводит полученный результат.

Нейросеть, в которой есть только линейные слои и различные функции активации – **полносвязные (fully connected) нейронные сети** или **многослойный перцептрон (multilayer perceptron)**.

## Глава 5

# Основные алгоритмы

### 5.1 Backpropagation

#### 5.1.1 Вступление

Алгоритмически описать данный процесс достаточно просто:

- Вычислить функцию потерь  $C(w)$
- Вычислить градиент функции  $C(w)$ , учитывая все веса ( $w$ ) и биасы ( $b$ ) в нейронной сети
- Изменить веса ( $w$ ) и биасы ( $b$ ) пропорционально значению их градиента

Намного сложнее разобраться в том, как в действительности вычисляются эти градиенты.

#### 5.1.2 Почему вообще нужно задумываться о том, как работает ВР?

#### 5.1.3 Простейший пример

Рассмотрим следующую нейронную сеть:

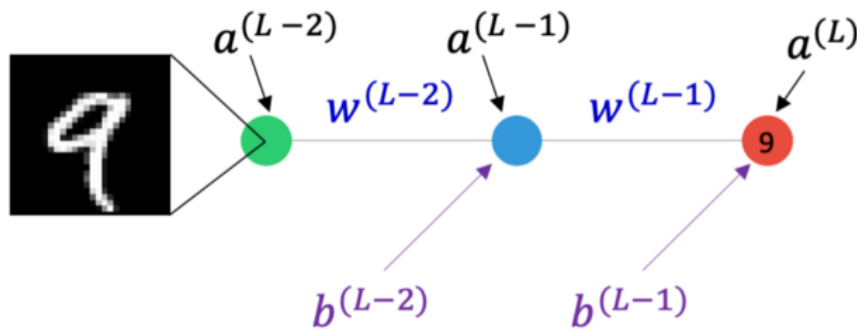


Рис. 5.1: простейшая нейронная сеть

На рисунке 5.1 изображена простейшая нейронная сеть, где зелёный, синий и красный нейрон отвечают за входной, скрытый и выходной нейрон (слой, если говорить в терминах более крупных нейронных сетей) соответственно. Обозначим функцию активации последнего нейрона как  $a^{(L)}$ , функцию активации предпоследнего слоя как  $a^{(L-1)}$ , а функцию активации первого слоя как  $a^{(L-2)}$ , где  $L$  - количество слоёв в нашей нейронной сети (в нашем случае  $L = 3$ ). Аналогично определим веса и биасы между слоем  $L - 1$  и  $L$  как  $w^{(L-1)}$  и  $b^{(L-1)}$ .

Представим, что наша картинка цифры 9 состоит из одного пикселя (понятно, что это не возможно; сделаем это для упрощения вычислений). Теперь, пусть после того, как этот один пиксель прошёл через все три слоя нашей простейшей нейронной сети (стадия прямого распространения), на выходе мы получили число 0.68 (так как мы решаем задачу классификации, то выходные значения должны находиться в диапазоне от 0 до 1, что можно интерпретировать как вероятность; для этого можно в качестве функции активации последнего слоя (в нашем случае нейрона) использовать функцию Softmax (добавить сюда потом ссылку)). Но наше желаемое выходное значение — это 1 (т.е. мы хотим (после завершения обучения), чтобы наша сеть была уверена на 100%, что данная картинка (в нашем случае один пиксель) — это число 9). Теперь, пусть мы будем использовать средне квадратичную ошибку в качестве функции потерь. Т.е. мы получаем, что потеря  $C$  для одного тренировочного образца (т.е. цифры 9) равняется  $(0.68 - 1)^2$

#### 5.1.4 Слой $L - 1$

Рассчитаем градиент нашей функции потерь  $C$  учитывая вес, соединяющий нейроны в слое  $L$  со слоем  $L - 1$ . Чтобы посмотреть, как бы мы это сделали, развернём последний слой нашей простейшей нейронной сети.

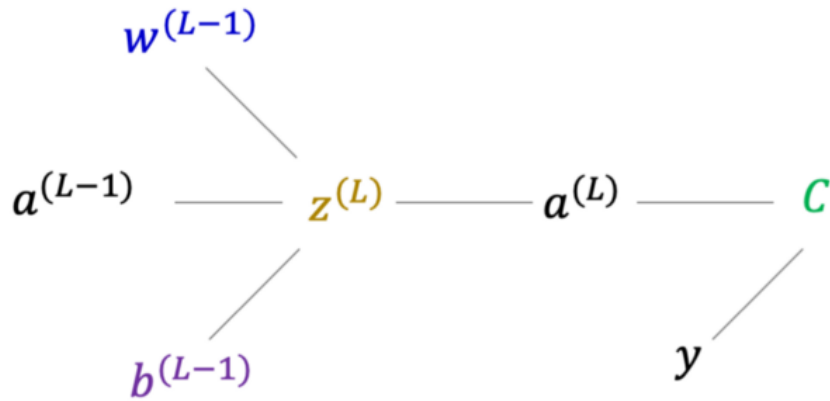


Рис. 5.2: Последний слой

Фигура 5.2 показывает, как каждый параметер последних двух слоёв влияет на потерю  $C$ . Чтобы получить взвешенную сумму  $z^{(L)}$  для последнего слоя, мы умножаем активацию слоя (т.е. выход предпоследнего слоя, к которому была применена соответствующая функция активации для данного слоя)  $L - 1$  на вес  $w^{(L-1)}$ , соединяющий два слоя, а затем мы прибавляем биас  $b^{(L-1)}$ . И наконец, мы пропускаем взвешенную сумму через нелинейную функцию (функцию активации)  $\sigma(z^{(L)})$ , чтобы посчитать  $a^{(L)}$ .

$$\begin{aligned}
 z^{(L)} &= w^{(L-1)} a^{(L-1)} + b^{(L-1)} \\
 a^{(L)} &= \sigma(z^{(L)})
 \end{aligned}$$

Рис. 5.3: Функция активации

Теперь мы хотим посмотреть, как сильно изменится  $C$ , если мы изменим

$w^{(L-1)}$ . Т.е. мы хотим посчитать  $\frac{\partial C}{\partial w^{(L-1)}}$ .

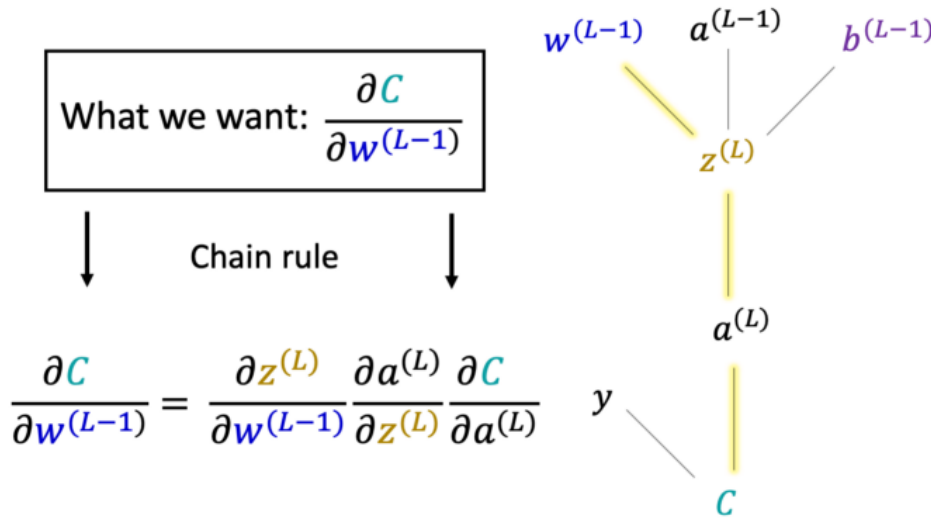


Рис. 5.4: Дифференцирование

Из картинки 5.4 можно понять, как вес  $w^{(L-1)}$  влияет на  $C$ . Сначала вес  $w^{(L-1)}$  даёт вклад во взвешенную сумму  $z^{(L)}$ , которая используется для того, чтобы посчитать потерю  $C$ . Найдём частную производную  $\frac{\partial C}{\partial w^{(L-1)}}$  явно.

$$\left. \begin{aligned} \frac{\partial C}{\partial a^{(L)}} &= 2(a^{(L)} - y) \\ \frac{\partial a^{(L)}}{\partial z^{(L)}} &= \sigma'(z^{(L)}) \\ \frac{\partial z^{(L)}}{\partial w^{(L-1)}} &= a^{(L-1)} \end{aligned} \right\} \quad \boxed{\frac{\partial C}{\partial w^{(L-1)}} = \frac{\partial z^{(L)}}{\partial w^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C}{\partial a^{(L)}} = a^{(L-1)} \sigma'(z^{(L)}) 2(a^{(L)} - y)}$$

Рис. 5.5: Находим частную производную явно

То, как получены последние два выражения в левой части - очевидно. Рассмотрим первое. Так как для подсчёта потери мы используем средне квадратичную ошибку (MSE, которая считается как  $(a^{(L)} - y)^2$ ), то если взять от этого производную по  $\partial a^{(L)}$ , то мы как раз получим  $2(a^{(L)} - y)$ .

Заметим, что мы не нашли производную от нашей функции активации (которую мы обозначили как  $\sigma$ ) явно. Если бы мы использовали Sigmoid в качестве нашей функции активации:

$$\frac{1}{1 + e^{-z}}$$

Тогда её производная равнялась бы:

$$\frac{e^{-z}}{(1 + e^{-z})^2}$$

**Замечание:** вот почему важно использовать в качестве функций активации всюду дифференцируемые функции, в противном случае мы не сможем рассчитать наши градиенты, однако функцию ReLu всё же можно использовать (добавить потом сюда ссылку).

Также мы хотим знать, как изменится значение нашей функции потерь, если мы изменим биас  $b^{(L-1)}$ .

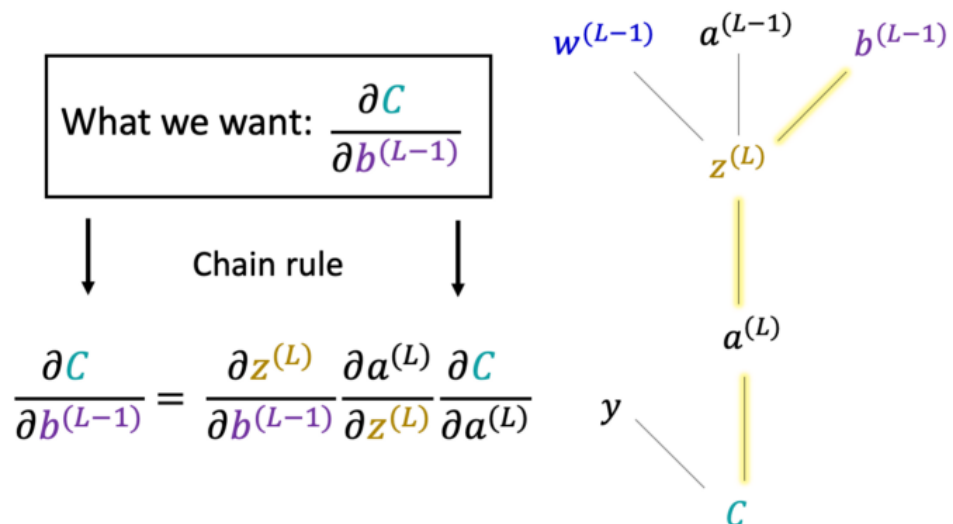


Рис. 5.6: Считаем производную для биаса

Как можно заметить, поменялся только первый множитель. Аналогично, найдём явную производную.



$$\left. \begin{aligned} \frac{\partial \mathcal{C}}{\partial a^{(L)}} &= 2(a^{(L)} - y) \\ \frac{\partial a^{(L)}}{\partial z^{(L)}} &= \sigma'(z^{(L)}) \\ \frac{\partial z^{(L)}}{\partial b^{(L-1)}} &= 1 \end{aligned} \right\} \boxed{\frac{\partial \mathcal{C}}{\partial b^{(L-1)}} = \frac{\partial z^{(L)}}{\partial b^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial \mathcal{C}}{\partial a^{(L)}} = \sigma'(z^{(L)}) 2(a^{(L)} - y)}$$

Рис. 5.7: Считаем производную для биаса

Отлично, теперь нейронная сеть может использовать полученные выражения для обновления весов и биасов по следующему принципу.

$$\begin{aligned} w^{(L-1)}(t+1) &= w^{(L-1)}(t) - \eta \frac{\partial \mathcal{C}}{\partial w^{(L-1)}} \\ b^{(L-1)}(t+1) &= b^{(L-1)}(t) - \eta \frac{\partial \mathcal{C}}{\partial b^{(L-1)}} \end{aligned}$$

Рис. 5.8: Обновление весов

На картинке 5.8  $\eta$  — это learning rate.

Несмотря на то, что нейронная сеть не имеет прямого контроля над  $a^{(L-1)}$ , скоро будет показано, что что нам будет это нужно, чтобы дальше двигаться по сети и обновлять параметры, поэтому проделаем предыдущие шаги, только для  $a^{(L-1)}$ .

Получаем:

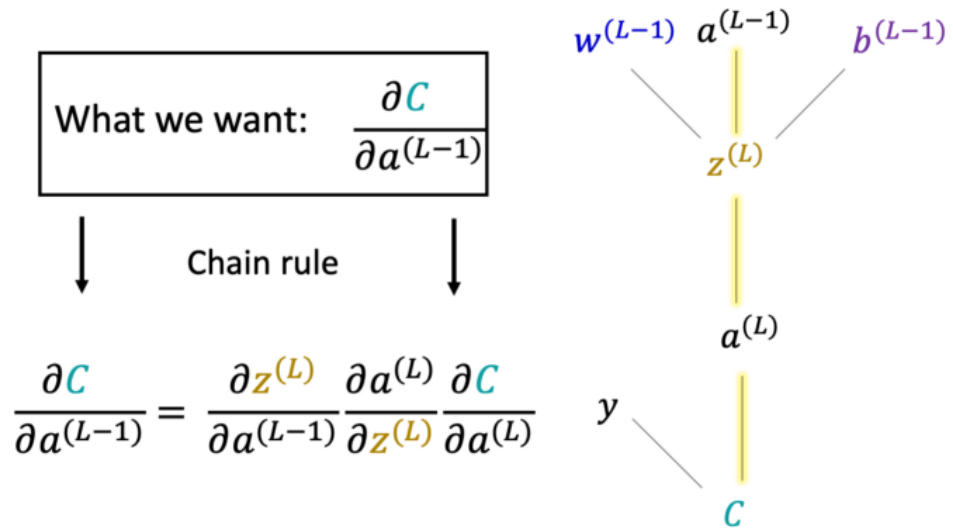


Рис. 5.9: Считаем градиент относительно функции активации предпоследнего слоя

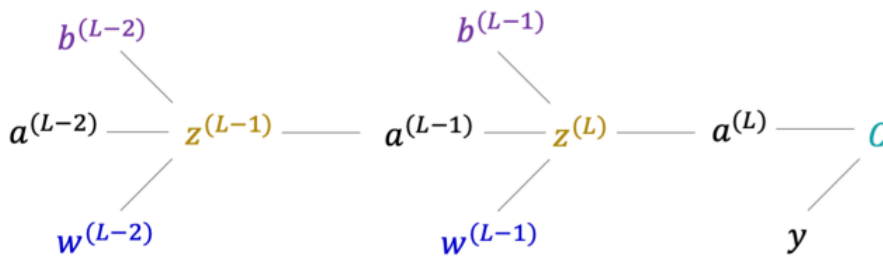
Находим производную явно:

$$\left. \begin{aligned} \frac{\partial C}{\partial a^{(L)}} &= 2(a^{(L)} - y) \\ \frac{\partial a^{(L)}}{\partial z^{(L)}} &= \sigma'(z^{(L)}) \\ \frac{\partial z^{(L)}}{\partial a^{(L-1)}} &= w^{(L-1)} \end{aligned} \right\} \quad \boxed{\frac{\partial C}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C}{\partial a^{(L)}} = w^{(L-1)} \sigma'(z^{(L)}) 2(a^{(L)} - y)}$$

Рис. 5.10: Находим производную явно

### 5.1.5 Слой $L - 2$

Теперь давайте посмотрим, как будет меняться значение функции потерь  $C$ , если изменять параметры в слое  $L - 2$ . Как и раньше, нарисует граф, чтобы наглядно продемонстрировать, как  $w^{(L-2)}$  и  $b^{(L-2)}$  косвенно влияют на значение потери  $C$ .

Рис. 5.11: Слои  $L - 1$  и  $L - 2$ 

Аналогично, используя цепное правило, посмотрим, как изменение веса  $w^{(L-2)}$  влияет на значение функции потерь  $C$ .

What we want:  $\frac{\partial C}{\partial w^{(L-2)}}$

↓ Chain rule ↓

$$\frac{\partial C}{\partial w^{(L-2)}} = \frac{\partial z^{(L-1)}}{\partial w^{(L-2)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \underbrace{\frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C}{\partial a^{(L)}}}_{\text{Chain rule}}$$

$$\frac{\partial C}{\partial w^{(L-2)}} = \frac{\partial z^{(L-1)}}{\partial w^{(L-2)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial C}{\partial a^{(L-1)}}$$

The diagram shows the backward pass for the weight  $w^{(L-2)}$ . It highlights the path from the cost  $C$  back to  $w^{(L-2)}$  through the layers  $L-1$  and  $L-2$ . The nodes  $z^{(L-1)}$ ,  $z^{(L)}$ ,  $a^{(L)}$ , and  $a^{(L-1)}$  are connected by yellow arrows, indicating the flow of gradients. The nodes  $w^{(L-2)}$ ,  $a^{(L-2)}$ ,  $b^{(L-2)}$ ,  $w^{(L-1)}$ ,  $a^{(L-1)}$ , and  $b^{(L-1)}$  are connected by grey arrows, indicating the forward pass.

Рис. 5.12: Частная производная для веса первого слоя

Теперь найдём производную явно:

Аналогичным образом можно посчитать градиенты для всех весов и биасов. Этот алгоритм называется **обратным распространением ошибки**.

$$\left. \begin{aligned} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} &= \sigma'(z^{(L-1)}) \\ \frac{\partial z^{(L-1)}}{\partial \mathbf{w}^{(L-2)}} &= \mathbf{a}^{(L-2)} \end{aligned} \right\} \boxed{\frac{\partial \mathcal{C}}{\partial \mathbf{w}^{(L-2)}} = \frac{\partial z^{(L-1)}}{\partial \mathbf{w}^{(L-2)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial \mathcal{C}}{\partial a^{(L-1)}} = \mathbf{a}^{(L-2)} \sigma'(z^{(L-1)}) \frac{\partial \mathcal{C}}{\partial a^{(L-1)}}}$$

Рис. 5.13: Производная в явном виде