# Методы машинного обучения

Шорохов С.Г.

кафедра информационных технологий

Лекция 4. Введение в глубокое обучение

# Глубокое обучение

Глубокое обучение представляет собой семество методов машинного обучения, основывающихся на искусственных нейронных сетях.

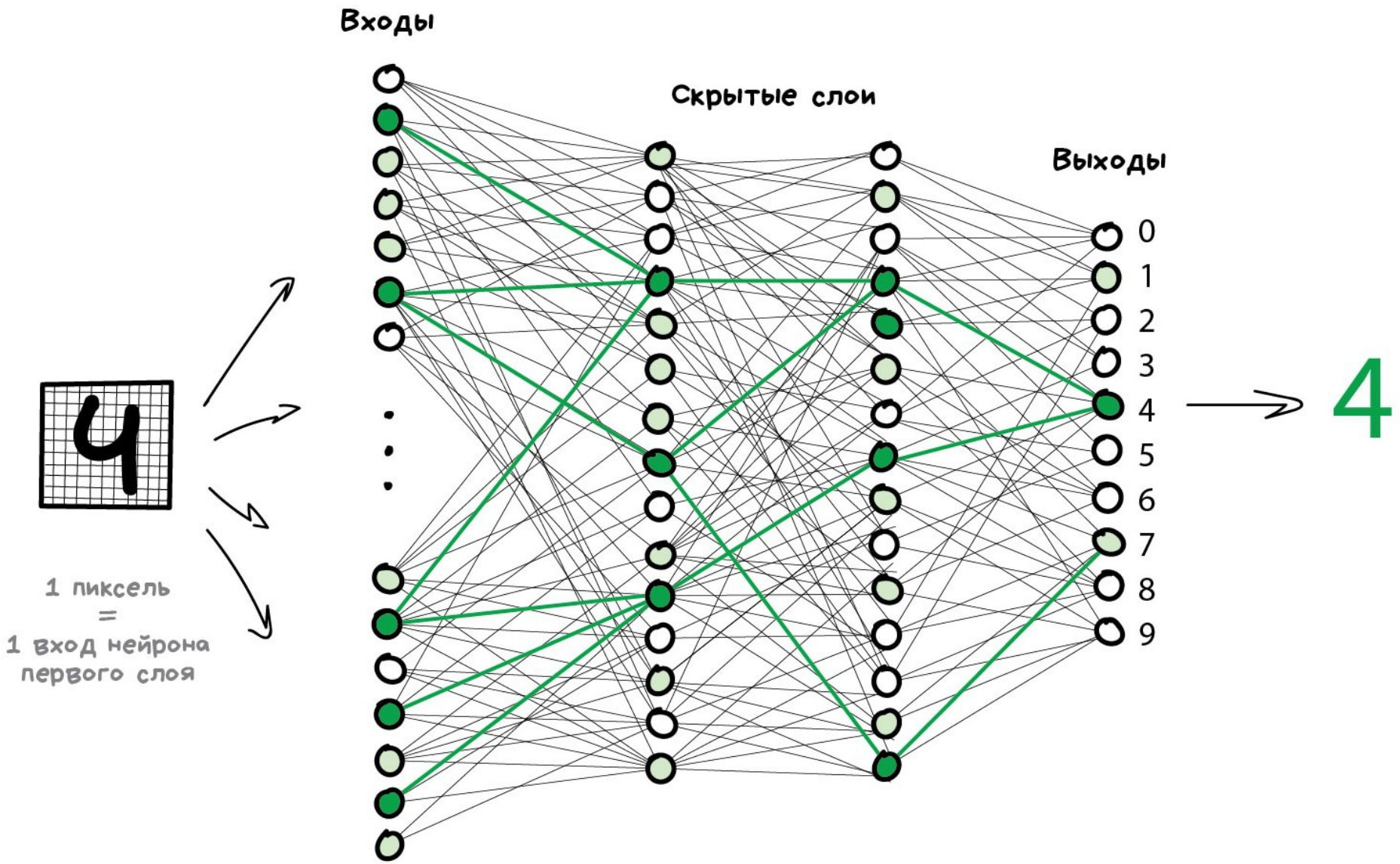
Математически искусственная нейронная сеть представляет собой направленный граф с нейронами в качестве вершин и связями между нейронами в виде ребер, причем вход для каждого нейрона является функцией взвешенной суммы выходов всех нейронов, связанных с ним входящими ребрами. Тогда выход нейронной сети равен

$$\mathbf{f}\left(\mathbf{x};\theta\right)=\psi_{\mathbf{d}}\left(...\psi_{\mathbf{2}}\left(\psi_{\mathbf{1}}\left(\mathbf{x}\right)\right)\right),\ \psi_{\mathbf{i}}\left(\mathbf{x}\right)=\sigma_{\mathbf{i}}\left(\mathbf{w}^{(\mathbf{i})}\,\mathbf{x}+\mathbf{b}^{(\mathbf{i})}\right)$$

Здесь каждый слой сети представляется функцией активации $\sigma_{\mathbf{i}}$ с аргументом в виде взвешенной суммы входных данных $\mathbf{x}$ с весами $\mathbf{w}^{(\mathbf{i})}$ и смещениями $\mathbf{b}^{(\mathbf{i})}$. Число слоев $\mathbf{d}$ называется глубиной нейронной сети и количество нейронов в слое представляет собой ширину этого слоя.

Целью глубокого обучения является определение набора параметров сети $\theta=\left\{\mathbf{w}^{(\mathbf{i})},\mathbf{b}^{(\mathbf{i})}\right\}_{\mathbf{i=1}}^{\mathbf{d}}$, который минимизирует функцию потерь $\mathcal{L}(\theta)$, определяющую качество модели при заданном наборе параметров $\theta$.

# Визуализация нейронной сети



# Теорема представления Колмогорова-Арнольда

А.Н.Колмогоров и В.И.Арнольд (в 1956–1958 гг) доказали, что если $\mathbf{f}$ – это многомерная непрерывная функция, то $\mathbf{f}$ можно записать в виде конечной композиции непрерывных функций одной переменной и бинарной операции сложения, а именно,

$$\mathbf{f}\left(\mathbf{x_1}, ..., \mathbf{x_n}\right) = \sum_{\mathbf{j=1}}^{\mathbf{2n+1}} \mathbf{g_j}\left(\sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{h_{ij}}\left(\mathbf{x_i}\right)\right),$$

где $\mathbf{g_j}$ и $\mathbf{h_{ij}}$ – непрерывные функции одной переменной и функции $\mathbf{h_{ij}}$ не зависят от выбора функции $\mathbf{f}$. Эта теорема тесно связана с т.н. 13-й проблемой Гильберта.

Теорема Колмогорова-Арнольда может быть интерпретирована, что любая непрерывная функция $\mathbf{n}$ переменных может быть представлена при помощи нейронной сети с двумя слоями.

# Универсальная теорема аппроксимации

Универсальная теорема аппроксимации (Дж. Цыбенко, 1989) утверждает, что искусственная нейронная сеть прямого распространения (feed-forward) с одним скрытым слоем может аппроксимировать любую непрерывную функцию многих переменных с любой точностью.

Более точно, если дана любая непрерывная функция действительных переменных $f$ на $[0,\,1]^n$ и $\varepsilon > 0$, то существуют векторы $\mathbf{w}_1, ..., \mathbf{w}_N$, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ и параметризованная функция $G\left(\cdot, \mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}\right) : [0,\,1]^n \to \mathbb{R}$ вида

$$G\left(\mathbf{x}, \mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}\right) = \sum_{i=1}^{N} \alpha_i \varphi\left(\mathbf{w}_i^T \mathbf{x} + \beta_i\right), \; \varphi\left(\xi\right) = \frac{1}{1 + e^{-\xi}},$$

такая, что для всех $x \in [0,\,1]^n$ выполняется условие

$$\left|G\left(\mathbf{x}, \mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}\right) - f\left(\mathbf{x}\right)\right| < \varepsilon,$$

$\mathbf{w} = \left(\mathbf{w}_1, ..., \mathbf{w}_N\right), \mathbf{w}_i \in \mathbb{R}^n$ – веса между входными нейронами и нейронами скрытого слоя, $\alpha = \left(\alpha_1, ..., \alpha_N\right)$ – веса между связями от нейронов скрытого слоя и выходным нейроном, $\boldsymbol{\beta} = \left(\beta_1, ..., \beta_N\right)$ – смещения для нейронов входного слоя.

# Функция потерь (ошибки)

В машинном обучении функция потерь необходима для количественной оценки ошибки между прогнозом и целевыми значениями.

Пусть рассматривается задача регрессии, т.е. прогнозирования значений зависимой переменной (отклика) $Y$ на основе значений независимых переменных (признаков) $X_1, X_2, ..., X_d$. В качестве функции потерь можно выбрать среднеквадратическую ошибку (MSE)

$$\mathcal{L}_2(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|^2,$$

где $n$ — количество точек данных, а $\mathbf{y} = (y_1, ..., y_n)$ и $\hat{\mathbf{y}} = (\hat{y}_1, ..., \hat{y}_n)$ — целевые значения и прогноз зависимой переменной соответственно. Средняя абсолютная ошибка (MAE) также может использоваться в качестве функции потерь для задач регрессии:

$$\mathcal{L}_1(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|.$$

Для задач классификации рассматриваются другие функции потерь (например, кросс-энтропия).

# Градиентный спуск (пакетный градиентный спуск)

Пусть объекты задаются $d$ числовыми признаками $X_j$, $j = \overline{1,d}$ и пространство описаний признаков представляет собой $X = \mathbb{R}^d$.

Пусть алгоритм имеет параметры $\mathbf{w}$ и выбрана некоторая функция потерь $\mathcal{L}$. Для $i$-го объекта выборки и алгоритма с параметрами $\mathbf{w}$ обозначим функцию потерь через $\mathcal{L}_i(\mathbf{w})$. Минимизируем эмпирический риск $Q(\mathbf{w})$, являющийся оценкой математического ожидания функции потерь

$$\min_{\mathbf{w}} Q(\mathbf{w}) = \min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_i(\mathbf{w})$$

Если функция потерь принадлежит классу $C^1(X)$, то можно применить метод (пакетного) градиентного спуска. Выберем начальное приближение вектора весов $\mathbf{w}^{(0)}$ и каждый следующий вектор весов будем вычислять как

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \sum_{i=1}^{n} \nabla_{\mathbf{w}} \mathcal{L}_i\left(\mathbf{w}^{(t)}\right),$$

где $\eta$ – шаг градиента, смысл которого заключается в том, насколько сильно менять вектор весов в направлении, противоположном градиенту. Остановка алгоритма будет определяться сходимостью $Q(\mathbf{w})$ или $\mathbf{w}$.

# Алгоритм пакетного градиентного спуска

Входными данными для алгоритма пакетного градиентного спуска являются матрица входных данных $\mathbf{D} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$, функция потерь $\mathcal{L}(\mathbf{x}, \mathbf{w})$, шаг обучения $\eta > 0$, требуемая точность $\varepsilon > 0$.

Batch Gradient Descent $(\mathbf{D}, \mathcal{L}, \eta, \varepsilon)$:

$t \leftarrow 0$ // инициализировать счетчик шагов/итераций
$\mathbf{w}^{(0)} \leftarrow$ случайный вектор в $\mathbb{R}^d$ // начальный вектор весов
repeat
  $\nabla_{\mathbf{w}}^{(t)} \leftarrow 0$ // начальное значение градиента для весов $\mathbf{w}^{(t)}$
  foreach $i = 1, 2, \cdots, n$ do
    // добавить значение градиента в точке $\mathbf{x}_i$
    $\nabla_{\mathbf{w}}^{(t)} \leftarrow \nabla_{\mathbf{w}}^{(t)} + \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{x}_i, \mathbf{w}^{(t)})$
  // вычислили значение градиента $\mathcal{L}$ для $\mathbf{w}^{(t)}$ по всем точкам $\mathbf{x}_i$
  $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \cdot \nabla_{\mathbf{w}}^{(t)}$ // обновить оценку для весов
  $t \leftarrow t + 1$ // увеличить счетчик шагов/итераций
until $\left\|\mathbf{w}^{(t)} - \mathbf{w}^{(t-1)}\right\| \leqslant \varepsilon$

Algorithm 1: Пакетный градиентный спуск

# Стохастический градиентный спуск

Проблема пакетного градиентного спуска заключается в том, что для определения нового приближения вектора весов необходимо вычислить градиент для каждого элемента данных, что может сильно замедлять вычисления. Идея стохастического градиентного спуска заключается в ускорении алгоритма за счет использования только одного элемента. То есть теперь новое приближение будет вычисляться как

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla_{\mathbf{w}} \mathcal{L}_i \left( \mathbf{w}^{(t)} \right),$$

где $i$ — случайно выбранный индекс точки набора данных. Если для останова алгоритма используется эмпирический риск $Q(\mathbf{w})$, то подсчет эмпирического риска $Q(\mathbf{w})$ на каждом шаге является слишком дорогостоящим, поэтому чтобы ускорить оценку $Q(\mathbf{w})$, можно использовать одну из приближенных рекуррентных формул ($\varepsilon_t = \mathcal{L}_i \left( \mathbf{w}^{(t)} \right)$):

- среднее арифметическое: $\overline{Q}_t = \frac{1}{t} (\varepsilon_t + \varepsilon_{t-1} + ...) = \frac{1}{t} \varepsilon_t + \left(1 - \frac{1}{t}\right) \overline{Q}_{t-1}$
- экспоненциальное скользящее среднее: $\overline{Q}_t = \lambda \varepsilon_t + (1 - \lambda) \overline{Q}_{t-1}$, где $\lambda$ – темп забывания предыстории значений $Q(\mathbf{w})$.

# Алгоритм стохастического градиентного спуска

Входными данными для алгоритма стохастического градиентного спуска являются матрица данных $\mathbf{D} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$, функция потерь $\mathcal{L}(\mathbf{x}, \mathbf{w})$, шаг обучения $\eta > 0$, требуемая точность $\varepsilon > 0$.

Stochastic Gradient Descent $(\mathbf{D}, \mathcal{L}, \eta, \varepsilon)$:

1 $t \leftarrow 0$ // инициализировать счетчик шагов/итераций
2 $\mathbf{w}^{(0)} \leftarrow$ случайный вектор в $\mathbb{R}^d$ // начальный вектор весов
3 repeat
4  foreach $i = 1, 2, \cdots, n$ (в случайном порядке) do
   // обновить оценку для весов по градиенту в точке $\mathbf{x}_i$
5  $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \cdot \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{x}_i, \mathbf{w}^{(t)})$
6  $t \leftarrow t + 1$ // увеличить счетчик шагов/итераций
7 until $\left\|\mathbf{w}^{(t)} - \mathbf{w}^{(t-1)}\right\| \leqslant \varepsilon$

Algorithm 2: Стохастический градиентный спуск

# Мини-пакетный градиентный спуск

Мини-пакетный (mini-batch) градиентный спуск – это разновидность алгоритма градиентного спуска, в которой обучающий набор данных разбивается на небольшие партии (мини-пакеты), которые используются для обновления коэффициентов (весов) модели и расчета ошибки модели. То есть каждый следующий вектор весов будет вычисляться как

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \sum_{i \in \mathbf{I}_t} \nabla_{\mathbf{w}} \mathcal{L}_i \left( \mathbf{x}_i, \mathbf{w}^{(t)} \right),$$

где $\mathbf{I}_t \subset \{1, ..., n\}$ – множество индексов точек в мини-пакете для эпохи обучения $t$, $\eta$ – шаг градиента. Мини-пакетный градиентный спуск стремится найти баланс между надежностью стохастического градиентного спуска и эффективностью пакетного градиентного спуска. Это наиболее распространенная реализация градиентного спуска, используемая в глубоком обучении.

Мини-пакетный градиентный спуск обеспечивает вычислительно более эффективный процесс, чем стохастический градиентный спуск. Пакетирование позволяет эффективно использовать ресурсы и реализовывать параллельные алгоритмы машинного обучения.

# Алгоритм мини-пакетного градиентного спуска

Входными данными для алгоритма мини-пакетного градиентного спуска являются матрица данных $\mathbf{D} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$, функция потерь $\mathcal{L}(\mathbf{x}, \mathbf{w})$, размер мини-пакета $m$, шаг обучения $\eta > 0$, требуемая точность $\varepsilon > 0$.

Mini-Batch Gradient Descent $(\mathbf{D}, \mathcal{L}, m, \eta, \varepsilon)$:

$t \leftarrow 0$ // инициализировать счетчик эпох
$\mathbf{w}^{(0)} \leftarrow$ случайный вектор в $\mathbb{R}^d$ // начальный вектор весов
repeat
  $\nabla_{\mathbf{w}}^{(t)} \leftarrow 0$ // начальное значение градиента для весов $\mathbf{w}^{(t)}$
  $\mathbf{I}_t \leftarrow$ случайный набор из $m$ индексов от 1 до $n$
  foreach $i \in \mathbf{I}_t$ do
    $\nabla_{\mathbf{w}}^{(t)} \leftarrow \nabla_{\mathbf{w}}^{(t)} + \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{x}_i, \mathbf{w}^{(t)})$ // градиент в точке $\mathbf{x}_i$
  // вычислили градиент $\mathcal{L}$ для $\mathbf{w}^{(t)}$ по точкам мини-пакета
  $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \cdot \nabla_{\mathbf{w}}^{(t)}$ // обновить оценку для весов
  $t \leftarrow t + 1$ // увеличить счетчик эпох
until $\left\|\mathbf{w}^{(t)} - \mathbf{w}^{(t-1)}\right\| \leqslant \varepsilon$

Algorithm 3: Мини-пакетный градиентный спуск

# Параметры алгоритмов градиентного спуска

Инициализация весов:

- $\mathbf{w}^{(0)} = \mathbf{0}$(кроме глубокого обучения)
- $w_j^{(0)} = random\left(-\frac{1}{2n}, \frac{1}{2n}\right)$
- $w_j^{(0)} = \frac{\mathbb{E}[y\, X_j]}{\mathbb{E}\left[X_j^2\right]}$, где $y$ – метки классов (для задачи классификации)

Случайный выбор точек набора данных:

- выбор элементов из разных классов
- выбор элементов, на которых ошибка больше

Выбор величины шага градиента:

- $\eta_t = \frac{1}{t}$
- метод скорейшего градиентного спуска: $\min_\eta \mathcal{L}_i\left(\mathbf{w} - \eta \nabla_{\mathbf{w}} \mathcal{L}_i\left(\mathbf{w}\right)\right)$
- при квадратичной функции потерь можно использовать $\eta = \left\|\mathbf{x}_i\right\|^2$
- иногда можно выполнять случайные шаги, чтобы выбивать процесс из локальных минимумов
- метод Левенберга-Марквардта (комбинация градиентного спуска и метода Гаусса — Ньютона)

# Автоматическое дифференцирование

Автоматическое дифференцирование (automatic differentiation, AD) представляет собой набор методов для оценки производной функции, заданной компьютерной программой. AD использует тот факт, что каждая компьютерная программа, какой бы сложной она ни была, выполняет последовательность элементарных арифметических операций (сложение, вычитание, умножение, деление и т. д.) и элементарных функций (exp, log, sin, cos и т. д.). При многократном применении цепного правила производные произвольного порядка могут быть вычислены автоматически с приемлемой точностью.

Автоматическое дифференцирование отличается от символьного дифференцирования и численного дифференцирования. Символьное дифференцирование сталкивается с трудностями преобразования алгоритма в одно математическое выражение и может привести к неэффективному компьютерному коду. Численное дифференцирование (метод конечных разностей) может вносить ошибки округления в процессе дискретизации. Наконец, оба этих классических метода работают медленно при вычислении частных производных функции по множеству входных данных, что необходимо для алгоритмов оптимизации на основе градиента.

# Цепное правило

В основе AD лежит т.н. цепное правило (правило дифференцирования сложной функции). Для сложной функции:

$$y = f(g(h(x))) = f(g(h(w_0))) = f(g(w_1)) = f(w_2) = w_3,$$

где $w_0 = x$, $w_1 = h(w_0)$, $w_2 = g(w_1)$, $w_3 = f(w_2) = y$, цепное правило означает

$$\frac{dy}{dx} = \frac{dy}{dw_2}\frac{dw_2}{dw_1}\frac{dw_1}{dx} = \frac{df(w_2)}{dw_2}\frac{dg(w_1)}{dw_1}\frac{dh(w_0)}{dx}$$

Обычно рассматривают два различных режима AD: прямое дифференцирование (или прямой режим), при котором цепное правило проходится изнутри наружу (т. е. сначала вычисляется $dw_1/dx$ , а затем $dw_2/dw_1$ и, наконец, $dy/dw_2$), и обратное дифференцирование (или обратный режим), при котором происходит обход снаружи внутрь (сначала вычисляют $dy/dw_2$, затем $dw_2/dw_1$ и, наконец, $dw_1/dx$. То есть:

1. прямое дифференцирование: $\frac{dw_i}{dx} = \frac{dw_i}{dw_{i-1}}\frac{dw_{i-1}}{dx}$ при $w_0 = x$
2. обратное дифференцирование: $\frac{dy}{dw_i} = \frac{dy}{dw_{i+1}}\frac{dw_{i+1}}{dw_i}$ при $w_3 = y$

# Прямое дифференцирование

При прямом дифференцировании сначала фиксируется независимая переменная, по которой выполняется дифференцирование, и рекурсивно вычисляется производная каждого подвыражения. В ручном расчете это включает в себя повторную замену производной внутренних функций в цепном правиле:

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial w_{n-1}} \frac{\partial w_{n-1}}{\partial x} = \frac{\partial y}{\partial w_{n-1}} \left( \frac{\partial w_{n-1}}{\partial w_{n-2}} \frac{\partial w_{n-2}}{\partial x} \right) = \frac{\partial y}{\partial w_{n-1}} \left( \frac{\partial w_{n-1}}{\partial w_{n-2}} \left( \frac{\partial w_{n-2}}{\partial w_{n-3}} \frac{\partial w_{n-3}}{\partial x} \right) \right) = \dots$$

Эту формулу можно обобщить на несколько переменных при помощи матричного произведения якобианов.

По сравнению с обратным дифференцированием прямое дифференцировании является естественным и простым в реализации, поскольку поток данных с производными совпадает с порядком вычислений. Каждая переменная $w$ дополняется своей производной $\dot{w}$ (хранится как числовое значение, а не как символьное выражение):

$$\dot{w} = \frac{\partial w}{\partial x}$$

Затем производные вычисляются синхронно согласно шагам, указанным выше, и объединяются с другими производными с помощью цепного правила.

# Прямое дифференцирование – пример

В качестве примера рассмотрим функцию:

$$y = f(x_1, x_2) = x_1 x_2 + \sin x_1 = w_1 w_2 + \sin w_1 = w_3 + w_4 = w_5,$$

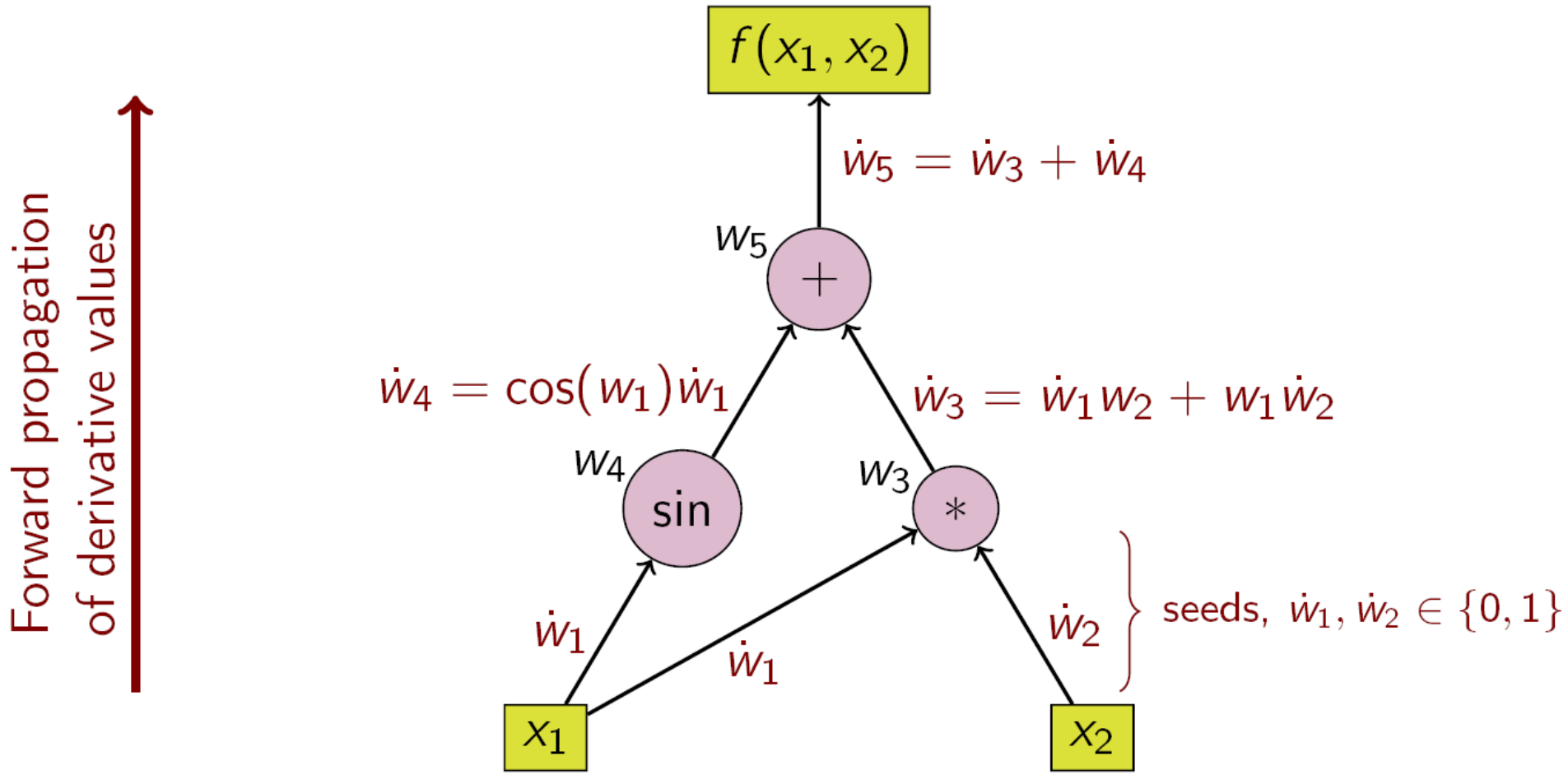
для которой $\frac{\partial y}{\partial x_1} = x_2 + \cos x_1$. Отдельные подвыражения в ходе вычисления функции помечены переменными $w_i$.

Выбор независимой переменной, по которой выполняется дифференцирование, влияет на начальные значения $\dot{w}_1$ и $\dot{w}_2$. Если требуется вычислить производную этой функции по $x_1$, то начальные значения должны быть следующими:

$$\dot{w}_1 = \frac{\partial x_1}{\partial x_1} = 1, \; \dot{w}_2 = \frac{\partial x_2}{\partial x_1} = 0$$

При заданных выше начальных значениях значения производных распространяются по вычислительному графу с использованием цепного правила. На следующем слайде показано графическое изображение этого процесса в виде вычислительного графа.

# Прямое дифференцирование – граф вычислений



# Прямое дифференцирование – вычисления

| Операции для вычисления значения функции | Операции для вычисления производной функции |
|---|---|
| $w_1 = x_1$ | $\dot{w}_1 = 1$ |
| $w_2 = x_2$ | $\dot{w}_2 = 0$ |
| $w_3 = w_1 w_2$ | $\dot{w}_3 = w_2 \dot{w}_1 + w_1 \dot{w}_2$ |
| $w_4 = \sin w_1$ | $\dot{w}_4 = \cos w_1 \dot{w}_1$ |
| $w_5 = w_3 + w_4$ | $\dot{w}_5 = \dot{w}_3 + \dot{w}_4$ |

$$\frac{\partial y}{\partial x_1} = \dot{w}_5 = \dot{w}_3 + \dot{w}_4 = w_2 \dot{w}_1 + w_1 \dot{w}_2 + \cos w_1 \dot{w}_1 = w_2 + \cos w_1 = x_2 + \cos x_1$$

Чтобы вычислить градиент этой функции, требуются производные от $f$ не только по $x_1$, но и по $x_2$, на вычислительном графе выполняется дополнительный цикл прямого дифференцирования с использованием начальных значений $\dot{w}_1 = 0$, $\dot{w}_2 = 1$.

Прямое дифференцирование более эффективно, чем обратное дифференцирование, для функций $f : \mathbb{R}^n \to \mathbb{R}^m$ для $m \gg n$, поскольку необходимо только $n$ проходов по графу по сравнению с $m$ проходами для обратного дифференцирования.

# Обратное дифференцирование

При обратном дифференцировании AD фиксируется зависимая переменная $y$, которую нужно дифференцировать, и рекурсивно вычисляется производная по каждому подвыражению. В ручном расчете производные внешних функций повторно подставляются в цепное правило:

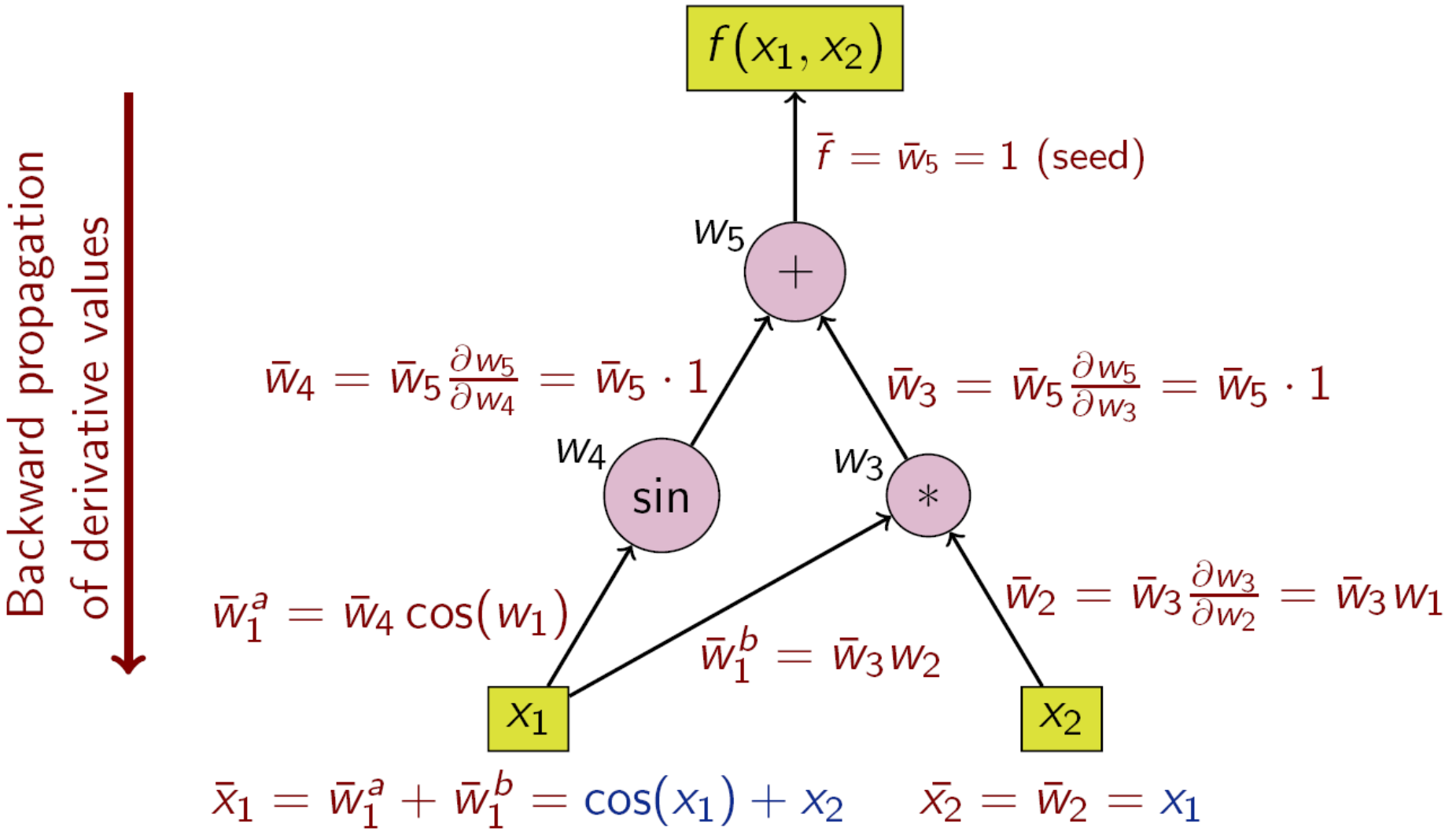
$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial w_1}\frac{\partial w_1}{\partial x} = \left(\frac{\partial y}{\partial w_2}\frac{\partial w_2}{\partial w_1}\right)\frac{\partial w_1}{\partial x} = \left(\left(\frac{\partial y}{\partial w_3}\frac{\partial w_3}{\partial w_2}\right)\frac{\partial w_2}{\partial w_1}\right)\frac{\partial w_1}{\partial x} = \ldots$$

При обратном дифференцировании интерес представляет т.н. присоединенная (adjoint) функция, обозначаемая $\bar{w}$. Это производная выбранной зависимой переменной $y$ по подвыражению $w$:

$$\bar{w} = \frac{\partial y}{\partial w}$$

Обратное дифференцирование происходит по цепному правилу снаружи внутрь или, в случае вычислительного графа на следующем слайде, сверху вниз.

# Обратное дифференцирование – граф вычислений



# Обратное дифференцирование – вычисления

В графе на предыдущем слайде происходило два прохода по вычислительному графу:

- прямой проход при вычислении значения функции с запоминанием промежуточных результатов
- обратный проход при вычислении значений производных функции

Функция в примере является скалярной, поэтому для вычисления производных используется только одно начальное значение, а для вычисления (двухкомпонентного) градиента требуется только один цикл прохода по вычислительному графу. Это только половина работы по сравнению с прямым дифференцированием, но обратное дифференцирование требует хранения промежуточных переменных $w_i$, а также инструкций для их вычисления в структуре данных, известной как «лента» (tape), которая может потреблять значительный объем памяти, если вычислительный граф большой. Это можно до некоторой степени смягчить, сохраняя только подмножество промежуточных переменных, а затем реконструируя необходимые рабочие переменные путем повторения вычислений. Для сохранения промежуточных состояний также можно использовать контрольные точки.

# Обратное дифференцирование – таблица

Операции для вычисления производной с использованием обратного дифференцирования показаны в таблице ниже (обратите внимание на обратный порядок при вычислении производной функции):

| Операции для вычисления значения функции | Операции для вычисления производной функции |
|---|---|
| $w_1 = x_1$ | $\bar{w}_5 = \frac{\partial y}{\partial w_5} = \frac{\partial y}{\partial y} = 1$ |
| $w_2 = x_2$ | $\bar{w}_4 = \frac{\partial y}{\partial w_4} = \frac{\partial y}{\partial w_5}\frac{\partial w_5}{\partial w_4} = \frac{\partial y}{\partial w_5} = \bar{w}_5$ |
| $w_3 = w_1 w_2$ | $\bar{w}_3 = \frac{\partial y}{\partial w_3} = \frac{\partial y}{\partial w_5}\frac{\partial w_5}{\partial w_3} = \frac{\partial y}{\partial w_5} = \bar{w}_5$ |
| $w_4 = \sin w_1$ | $\bar{w}_2 = \frac{\partial y}{\partial w_2} = \frac{\partial y}{\partial w_3}\frac{\partial w_3}{\partial w_2} = \frac{\partial y}{\partial w_3} w_1 = \bar{w}_3 w_1$ |
| $w_5 = w_3 + w_4$ | $\bar{w}_1 = \frac{\partial y}{\partial w_1} = \frac{\partial y}{\partial w_3}\frac{\partial w_3}{\partial w_1} + \frac{\partial y}{\partial w_4}\frac{\partial w_4}{\partial w_1} = \bar{w}_3 w_2 + \bar{w}_4 \cos w_1$ |

$$\frac{\partial y}{\partial x_1} = \bar{w}_3 w_2 + \bar{w}_4 \cos w_1 = \bar{w}_5 w_2 + \bar{w}_5 \cos w_1 = w_2 + \cos w_1 = x_2 + \cos x_1$$

# Обратное дифференцирование – вычисления

Графом потока данных вычисления функции можно манипулировать, чтобы вычислить градиент этой функции. Это делается путем добавления присоединенного узла для каждого основного узла, соединенного присоединенными ребрами, которые параллельны основным ребрам, но проходятся в противоположном направлении. Узлы в присоединенном графе представляют умножение на производные функций, вычисляемых узлами в основном графе.

Обратное дифференцирование более эффективно, чем прямое дифференцирование, для функций $f : \mathbb{R}^n \to \mathbb{R}^m$для $m \ll n$, поскольку необходимо только $m$ проходов по сравнению с $n$ проходами для прямого дифференцирования.

Обратный режим AD был впервые опубликован в 1976 году Сеппо Линнаинмаа (Seppo Linnainmaa).

Обратное распространение ошибок в нейронных сетях (многослойных персептронах) является частным случаем обратного режима AD (обратного дифференцирования).

# Проблема переобучения и недообучения

Одной из проблем машинного обучения в целом и глубокого обучения нейронных сетей в частности является т.н. переобучение.

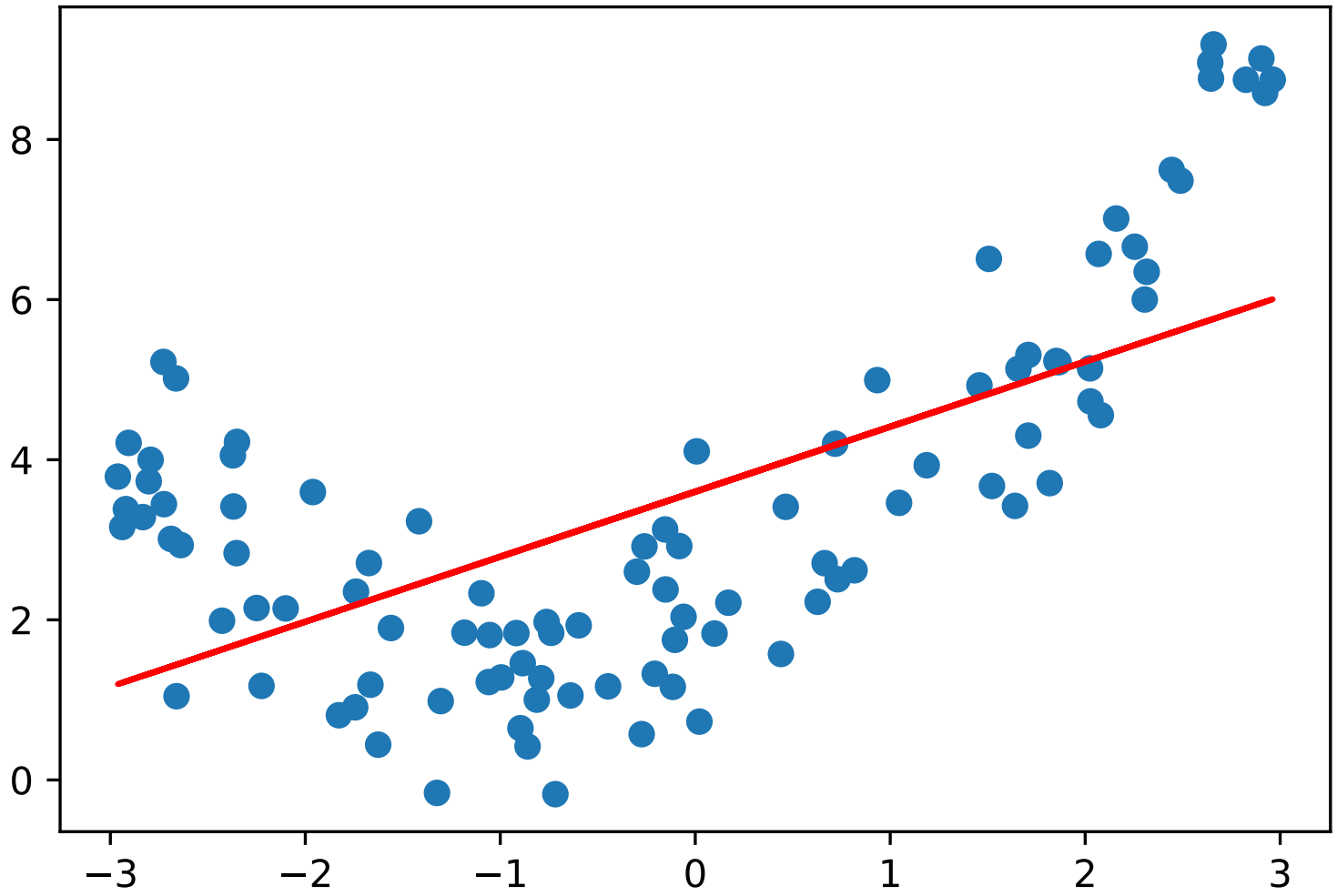
Переобучение (overfitting, overtraining) — это ситуация, когда обучаемая модель для данных обучающего набора прогнозирует отклики, которые близко или даже точно соответствуют откликам в этом наборе, однако для данных, не участвовавших в процессе обучения, модель вырабатывает предсказания низкого качества. Переобучение может возникать

- при использовании слишком сложных моделей
- при слишком долгом процессе обучения
- при неудачной обучающей выборке

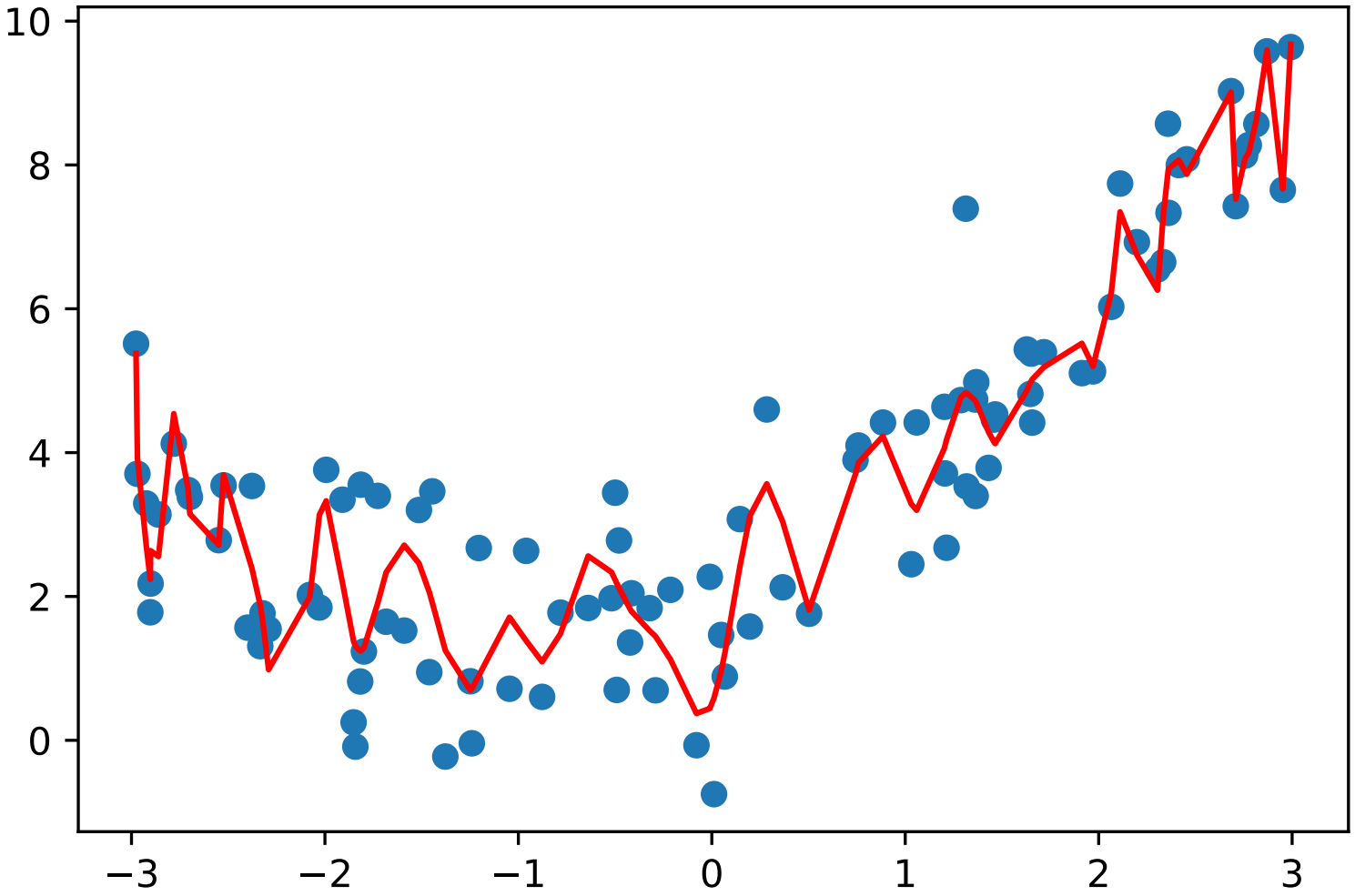
Недообучение (underfitting) — это ситуация, когда обучаемая модель не обеспечивает приемлемого качества (достаточно малой величины средней ошибки) даже на обучающем наборе. Недообучение может возникать

- при использовании слишком простых моделей
- при прекращении процесса обучения до достижения состояния с достаточно малой ошибкой
- при неудачной обучающей выборке

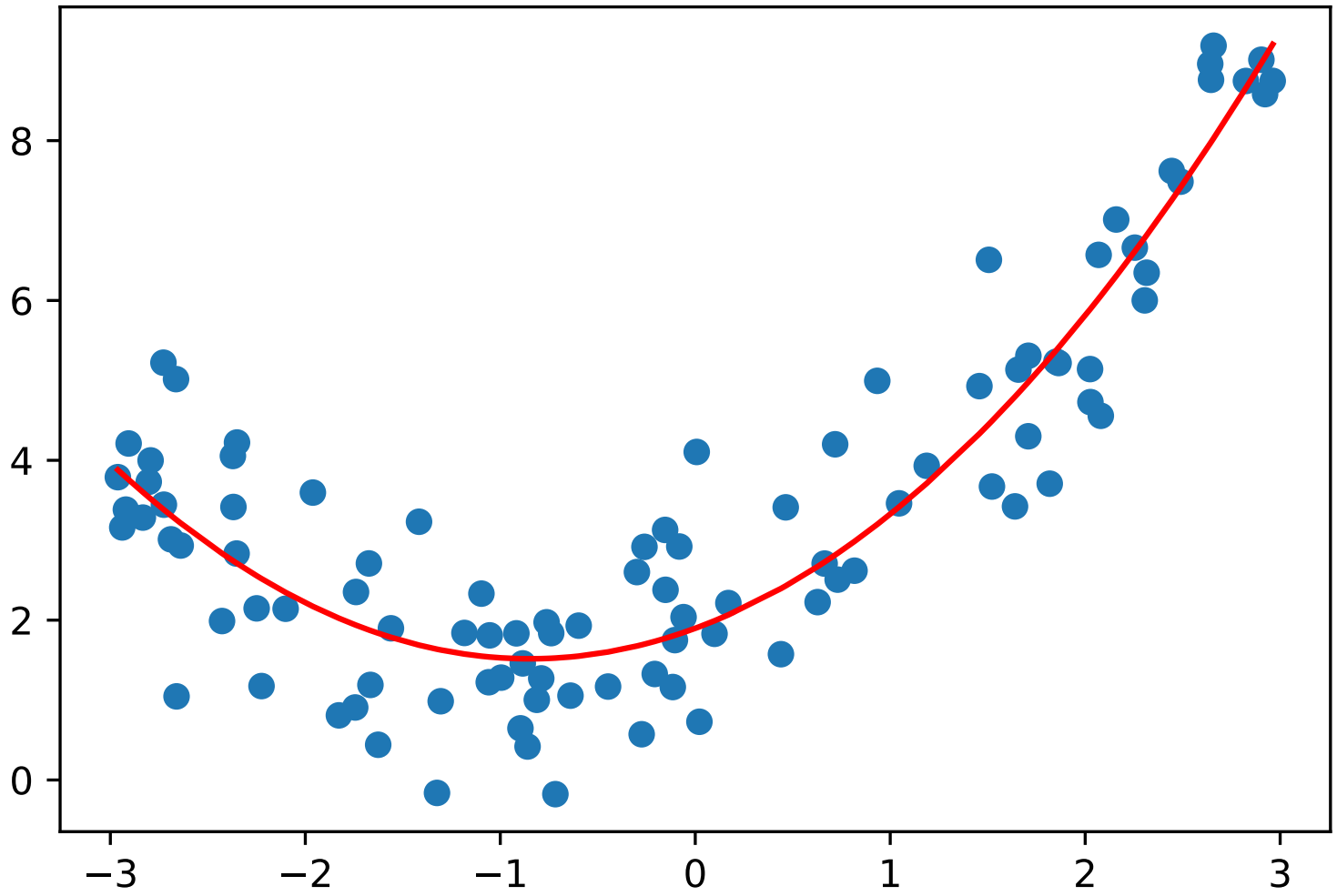
# Пример недообучения



# Пример переобучения



# Пример подходящей модели



# Переобучение нейронных сетей

Проблема переобучения в той или иной степени характерна для всех видов моделей машинного обучения, но особенно остро она стоит для нейронных сетей. В процессе обучения производится подгонка весов нейронной сети таким образом, что сеть преобразовывала входные данные к желаемым выходным данным в соответствии с зависимостями, обнаруженными в обучающих данных.

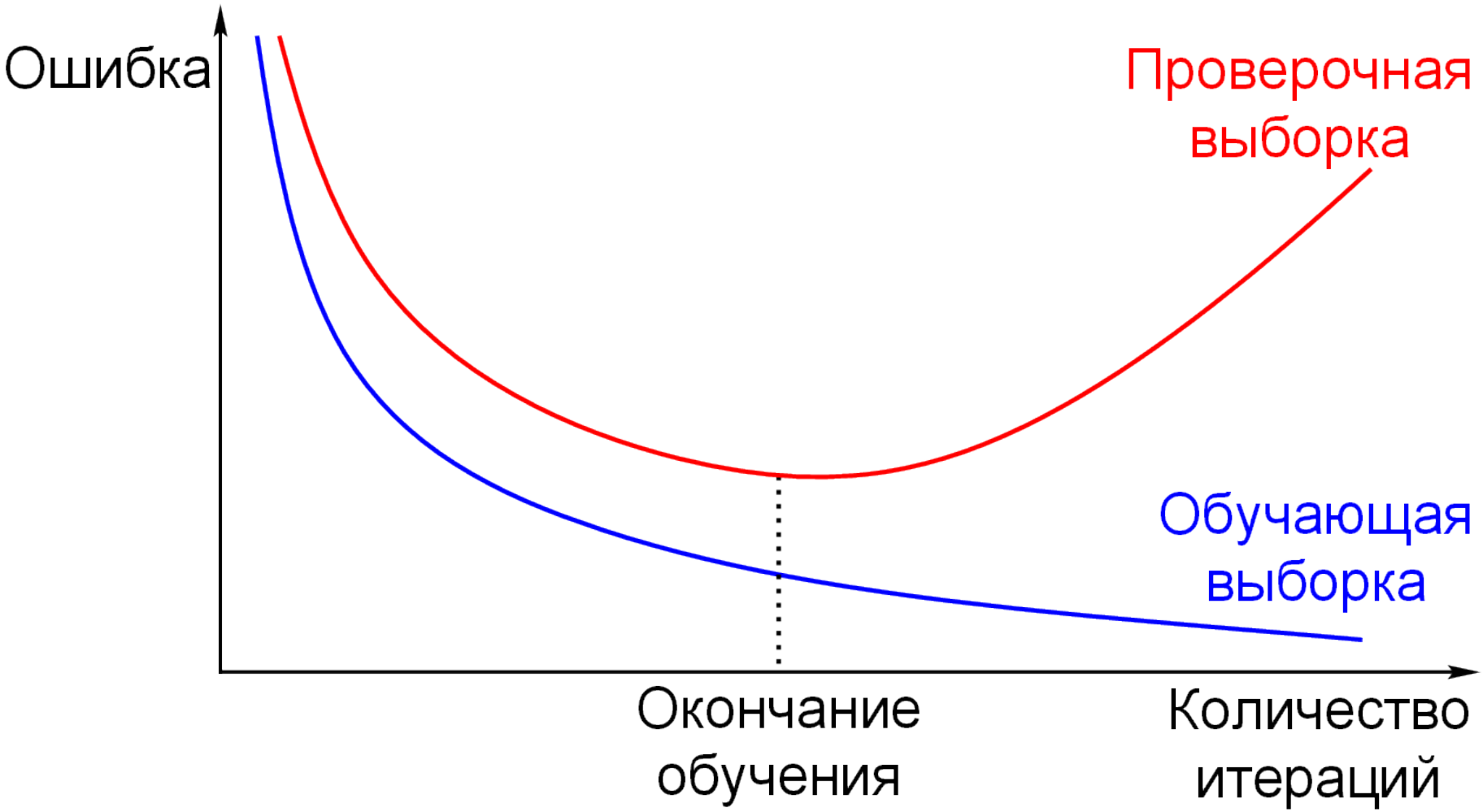
Однако, если нейронную сеть обучать слишком долго или с использованием слишком большого числа параметров (весов), то произойдёт переобучение. Это связано с тем, что с определённого момента сеть начинает «подстраиваться» не под общие зависимости в данных, а под особенности обучающего набора данных, который может содержать аномальные значения, ошибки и т.д.

Как следствие, нейронная сеть начнёт проверять новые, предъявляемые ей входные данные не на соответствие общим зависимостям, а на соответствие отдельным экземплярам данных из обучающего набора. В итоге модель сможет корректно предсказать отклик для новых данных только в том случае, если они совпадут с данными в обучающем наборе.

# Некоторые способы предотвращения переобучения

1. Применение тестовой выборки. Тестовая выборка формируется из набора данных случайным образом и применяется для оценки ошибки модели, но не влияет на корректировку весов сети. В начале обучения сети ошибка и на обучающей, и на тестовой выборках уменьшаются. Но начиная с какого-то момента ошибка на тестовой выборке начинает расти. Это сигнализирует о начале переобучения и необходимости принудительно остановить процесс обучения.
2. Использование кросс-валидации (перекрёстной проверки). Все данные, на которых строится модель, разделяются на $k$ блоков равного размера. При этом обучение производится на $k-1$ блоках, а тестирование — на $k$-м. Процедура повторяется $k$ раз, при этом для тестирования каждый раз выбирается другой блок. В результате все блоки оказываются используемыми и как обучающие, и как тестирующие.
3. Выбор конфигурации нейронной сети (числа слоёв и нейронов) так, чтобы количество параметров модели (весов) было в 2-3 раза меньше числа точек обучающей выборки. Если число параметров модели и точек данных окажется соизмеримым, то сеть запомнит все комбинации вход-выход в обучающих данных, и будет воспроизводить их, а на новых данных допускать ошибки.

# Останов обучения модели при переобучении



# Ожидаемые потери модели машинного обучения

Идеальный (оптимальный) алгоритм машинного обучения минимизирует функцию потерь. Поскольку истинное значение отклика $y$ для тестовой точки $\mathbf{x}$ неизвестно, целью обучения модели $M$ по набору данных $\mathbf{D}$ является минимизация ожидаемых потерь (по всем откликам):

$$\mathbb{E}_y \left[ \mathcal{L} \left( y, M \left( \mathbf{x} \right) \right) \mid \mathbf{x} \right] = \sum_y \mathcal{L} \left( y, M \left( \mathbf{x} \right) \right) \mathbb{P} \left[ y \mid \mathbf{x} \right],$$

где $\mathcal{L} \left( y, \hat{y} \right)$ – функция потерь (ошибка) для истинного отклика $y$ и прогноза $\hat{y}$, $\mathbb{P} \left[ y \mid \mathbf{x} \right]$ – это условная вероятность отклика $y$ для заданной тестовой точки $\mathbf{x}$, и $\mathbb{E}_y$ обозначает, что математическое ожидание вычисляется по различным откликам $y$. Ожидаемые потери для функции квадратичных потерь $\mathcal{L} \left( y, \hat{y} \right) = \left( y - \hat{y} \right)^2$ для заданной точки $\mathbf{x}$ и набора данных $\mathbf{D}$ можно разложить на смещение (bias, систематическую ошибку) и дисперсию (variance) следующим образом:

$$\mathbb{E}_y \left[ L \left( y, M \left( \mathbf{x}, \mathbf{D} \right) \right) \mid \mathbf{x}, \mathbf{D} \right] = \underbrace{\mathbb{E}_y \left[ \left( y - \mathbb{E}_y \left[ y \mid \mathbf{x} \right] \right)^2 \mid \mathbf{x}, \mathbf{D} \right]}_{var(y|\mathbf{x})} + \underbrace{\left( M \left( \mathbf{x}, \mathbf{D} \right) - \mathbb{E}_y \left[ y \mid \mathbf{x} \right] \right)^2}_{\text{квадратичная ошибка}}$$

Здесь $M \left( \mathbf{x}, \mathbf{D} \right)$ обозначает прогноз модели, обученной на наборе данных $\mathbf{D}$, для точки $\mathbf{x}$.

# Ожидаемые потери по всем обучающим выборкам

Для различных обучающих выборок $\mathbf{D}$ обученная модель будет иметь различный уровень ожидаемых потерь. Средняя или ожидаемая квадратичная ошибка для данной тестовой точки $\mathbf{x}$ по всем обучающим выборкам $\mathbf{D}$ тогда задается как

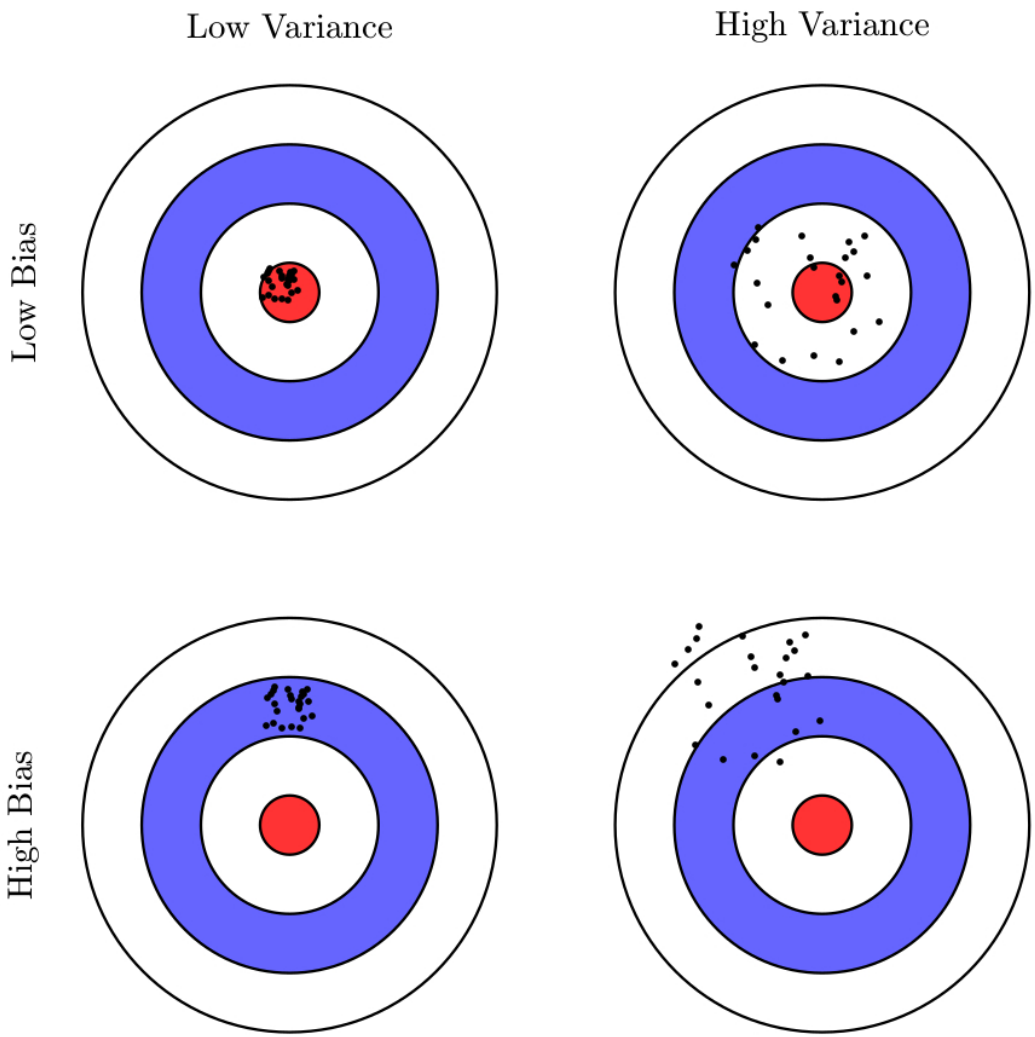
$$\mathbb{E}_{\mathbf{D}}\left[\left(M\left(\mathbf{x},\mathbf{D}\right)-\mathbb{E}_{y}\left[y\mid\mathbf{x}\right]\right)^{2}\right]=$$

$$=\underbrace{\mathbb{E}_{\mathbf{D}}\left[\left(M\left(\mathbf{x},\mathbf{D}\right)-\mathbb{E}_{\mathbf{D}}\left[M\left(\mathbf{x},\mathbf{D}\right)\right]\right)^{2}\right]}_{\text{дисперсия (variance)}}+\left(\underbrace{\mathbb{E}_{\mathbf{D}}\left[M\left(\mathbf{x},\mathbf{D}\right)\right]-\mathbb{E}_{y}\left[y\mid\mathbf{x}\right]}_{\text{смещение (bias)}}\right)^{2}$$

Здесь $\mathbb{E}_{\mathbf{D}}$ обозначает, что математическое ожидание вычисляется по различным обучающим выборкам $\mathbf{D}$.

Таким образом, высокие ожидаемые потери модели машинного обучения могут быть вызваны высоким смещением и/или высокой дисперсией модели.

# Смещение и дисперсия модели

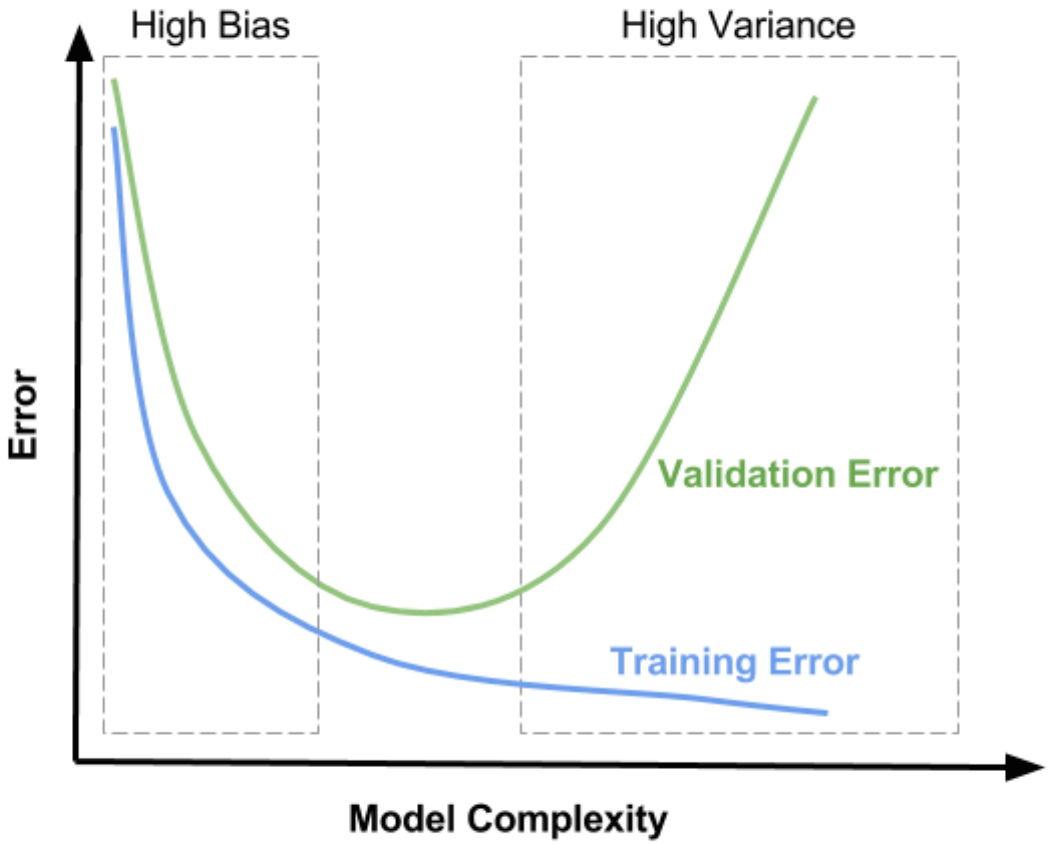



Смещение (bias) отражает ошибку, связанную с неверными допущениями модели машинного обучения. Высокое смещение может привести к недообучению модели.

Дисперсия (variance) отражает ошибку, вызванную большой чувствительностью модели к небольшим отклонениям в обучающем наборе. Высокая дисперсия может привести к переобучению модели.

# Конфликт между смещением и дисперсией

Конфликт между смещением и дисперсией (дилемма bias–variance) — это противоречие при попытке одновременно минимизировать смещение и дисперсию, тогда как уменьшение одного приводит к увеличению другого.




При небольшой сложности модели наблюдается высокое смещение. При усложнении модели смещение уменьшается, но дисперсия увеличивается и приводит к проблеме высокой дисперсии.

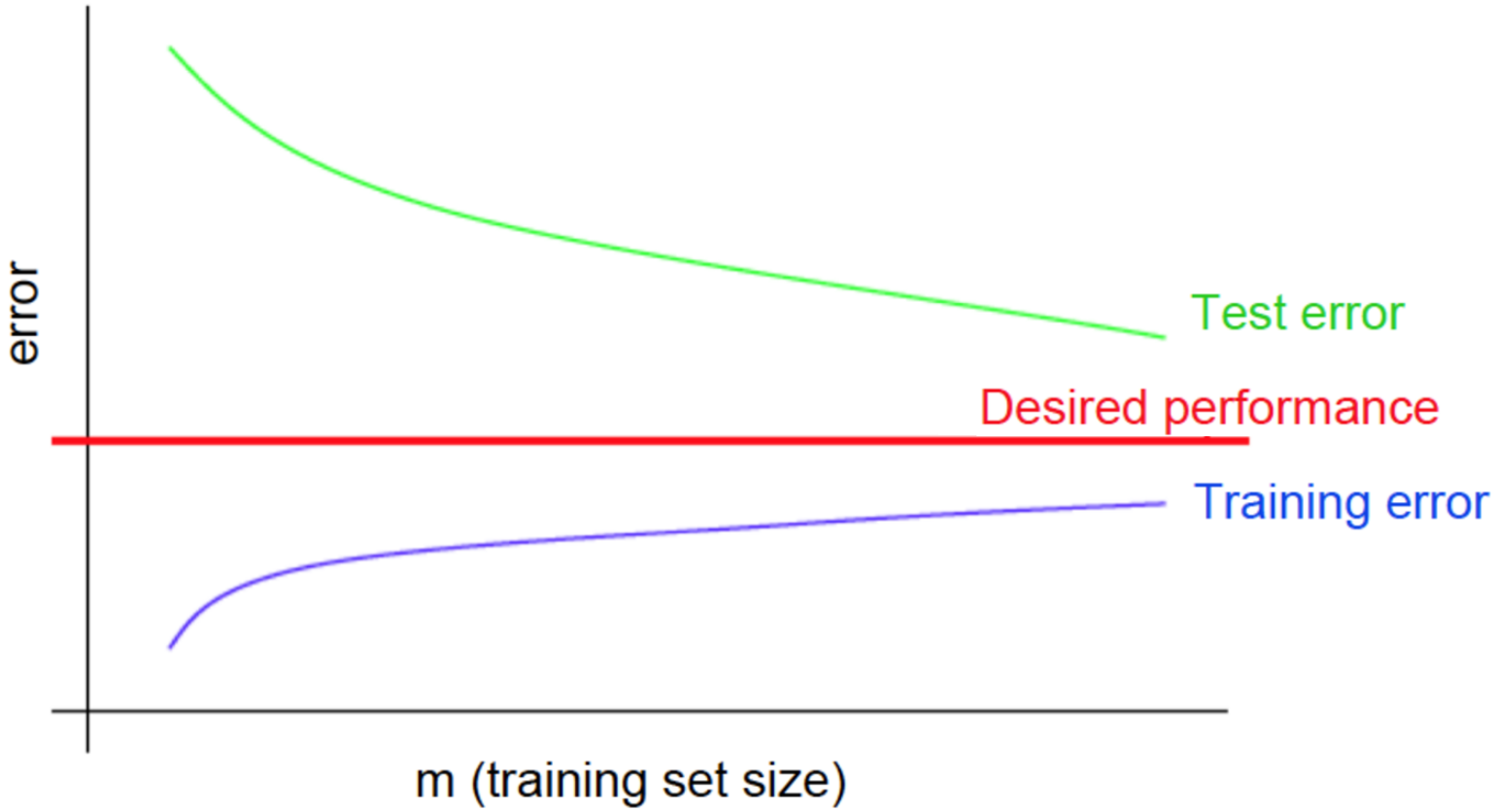
# Кривые обучения

Кроме графиков зависимости ошибки на обучающей и тестовой выборках в зависимости от количества эпох обучения для визуализации переобучения также используют т.н. кривые обучения.

Кривая обучения — графическое представление зависимости меры (показателя) качества обучения (по вертикальной оси) от определенного показателя модели обучения (по горизонтальной оси). Например, в примерах ниже представлена зависимость средней ошибки от размера обучающего набора данных.

# Кривые обучения при переобучении

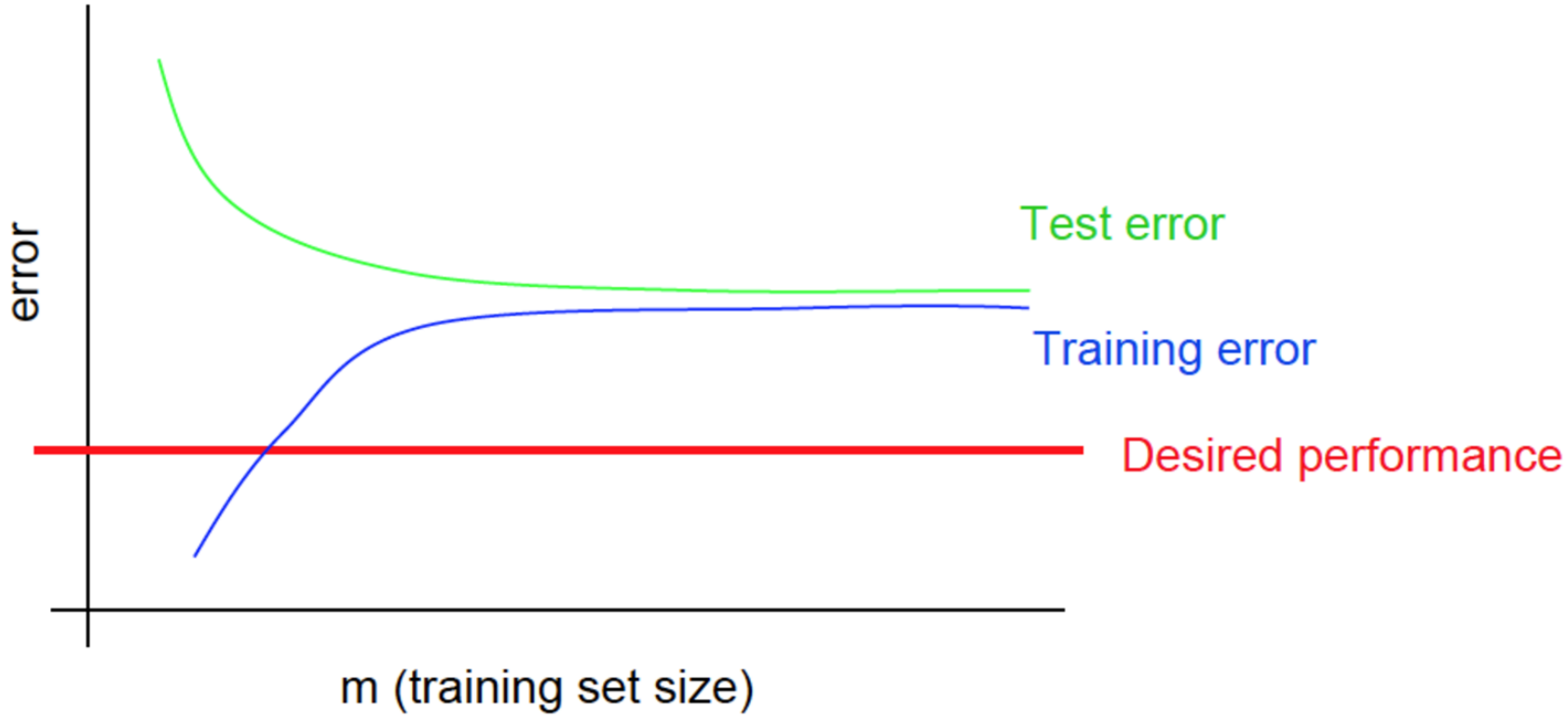
При переобучении небольшая средняя ошибка на обучающей выборке не обеспечивает такую же малую ошибку на тестовой выборке.




Переобучение связано с высокой дисперсией (variance) алгоритма обучения.

# Кривые обучения при недообучении

При недообучении независимо от объема обучающего датасета как на обучающей выборке, так и на тестовой выборке небольшая средняя ошибка не достигается.




Недообучение связано с высоким смещением (bias) алгоритма обучения.

# Регуляризация модели

Регуляризация — это метод добавления некоторых дополнительных ограничений к условиям модели (обычно в форме штрафа за сложность модели) с целью предотвратить переобучение модели.

Наиболее часто используемые виды регуляризации — $L_1$ и $L_2$ , а также их линейная комбинация.

Пусть $\mathcal{L}$ – функция потерь, а $\mathbf{w}$ — вектор параметров обучаемой модели $M(\mathbf{x}, \mathbf{w})$, а $\lambda$ — неотрицательный гиперпараметр, являющийся коэффициентом регуляризации. Будем минимизировать эмпирический риск $Q(\mathbf{w})$, являющийся оценкой математического ожидания функции потерь:

$$\min_{\mathbf{w}} Q(\mathbf{w}) = \min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(y_i,\ M(\mathbf{x}_i, \mathbf{w}))$$

Здесь $\mathbf{x}_i$ – точки обучающего набора данных $\mathbf{D} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$, $y_i$ – соответствующие отклики.

# $L_2$-регуляризация (ridge regularization)

$L_2$-регуляризация (регуляризация Тихонова, ridge regularization)

$$Q(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(y_i,\, M(\mathbf{x}_i, \mathbf{w})) + \lambda\sum_{j=1}^{d} w_j^2$$

Минимизация регуляризованного соответствующим образом эмпирического риска приводит к выбору такого вектора параметров $\mathbf{w}$, который не слишком сильно отклоняется от нуля. Это позволяет избежать переобучения.

$L_2$-регуляризация используется в т.н. гребневой регрессии (ridge regression).

# $L_1$-регуляризация (lasso regularization)

$L_1$-регуляризация (lasso regularization) или регуляризация через манхэттенское расстояние:

$$Q(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(y_i,\ M(\mathbf{x}_i, \mathbf{w})) + \lambda\sum_{j=1}^{d}|w_j|$$

Данный вид регуляризации также позволяет ограничить значения вектора $\mathbf{w}$. Однако, к тому же он обладает интересным и полезным на практике свойством — обнуляет значения некоторых параметров, что приводит к отбору признаков.

$L_1$-регуляризация используется в регрессии лассо (lasso regression).

# Регуляризация «эластичная сеть»

Эластичная сеть (elastic net regularization):

$$Q(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(y_i,\, M(\mathbf{x}_i, \mathbf{w})) + \lambda_1 \sum_{j=1}^{d}|w_j| + \lambda_2 \sum_{j=1}^{d} w_j^2$$

Эта регуляризация использует как $L_1$ , так и $L_2$ регуляризации, учитывая эффективность обоих методов. Ее полезной особенностью является то, что она создает условия для группового эффекта при высокой корреляции переменных, а не обнуляет некоторые из них, как в случае с $L_1$-регуляризацией.

# Решение проблем переобучения и недообучения

Возможные решения при переобучении:

- Увеличение количества данных в наборе;
- Уменьшение количества параметров модели;
- Добавление регуляризации (увеличение коэффициента регуляризации).

Возможные решения при недообучении:

- Добавление новых параметров в модель;
- Использование для описания модели более сложных функций (с большим числом параметров);
- Уменьшение коэффициента регуляризации.

# Аугментация данных

В машинном обучении иногда можно столкнуться с ситуацией, когда набор данных имеет ограниченный размер. Но чтобы получить лучшие результаты обобщения модели, необходимо иметь больше данных, в том числе и различные их вариации.

Аугментация данных (data augmentation) — это методика создания дополнительных данных из имеющихся данных.

Часто проблема ограниченного набора данных возникает при решении задач, связанных с обработкой изображений. Следующие способы аугментации изображений являются самыми популярными:

- Отображение по вертикали или горизонтали (flipping)
- Поворот изображения на определенный угол (rotation)
- Создание отступа (padding)
- Вырезание части изображения (cropping)
- Добавление шума (adding noise)
- Манипуляции с цветом (color jittering).

Также, можно применять различные комбинации, к примеру, вырезать часть изображения, повернуть его и изменить цвет фона.

# Сокращение весов (weight decay)

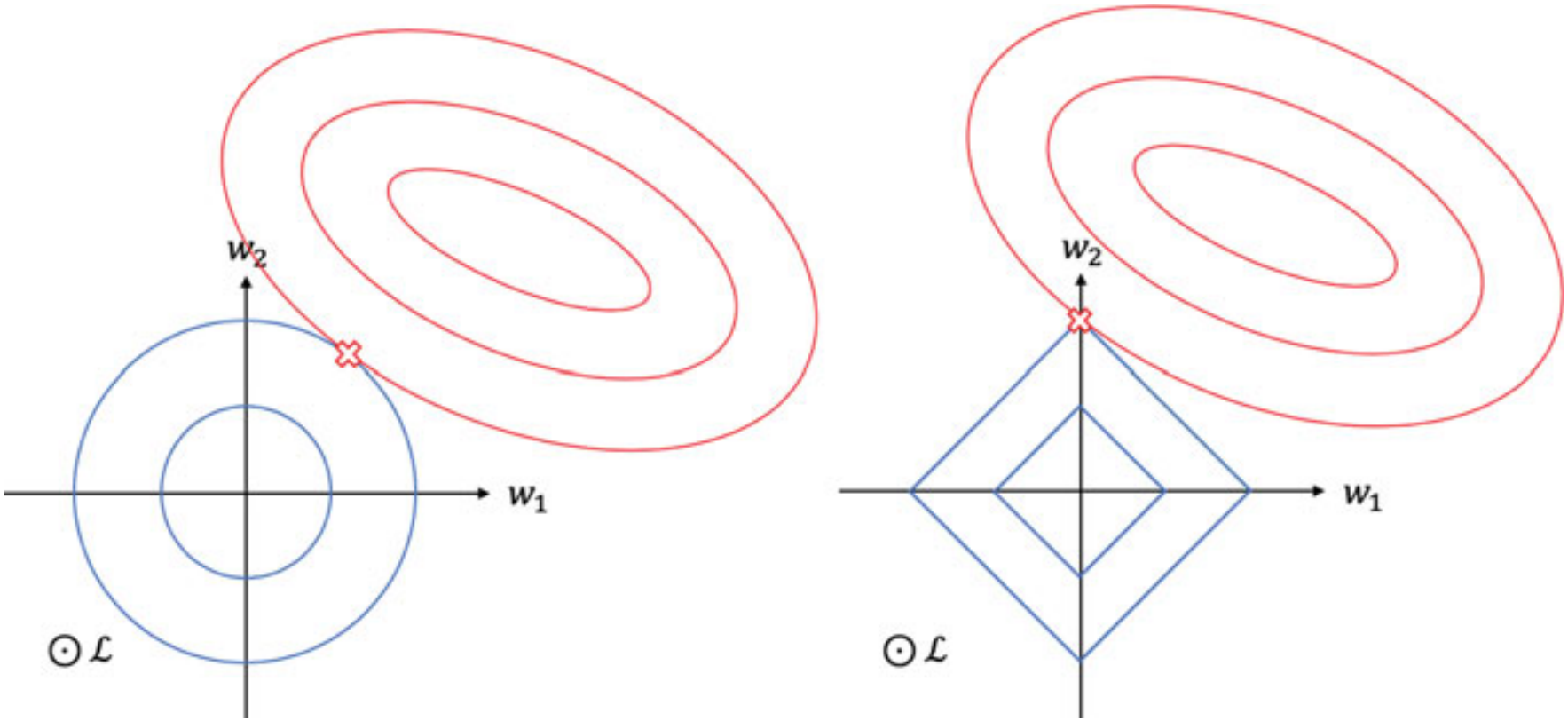
Сокращение весов (weight decay) – это простой, но эффективный метод регуляризации, направленный на решение проблемы переобучения нейронной сети. В функцию потерь в качестве штрафа вводится член регуляризации, поощряющий параметры $\mathbf{w}$ с меньшими абсолютными значениями. Функция потерь со штрафом за норму параметров $\mathbf{w}$ определяется следующим образом:

$$\mathcal{L}'(\mathbf{y}, \hat{\mathbf{y}}) = \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) + \lambda \Omega(\mathbf{w}),$$

где $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ – исходная функция потерь для целевых значений $\mathbf{y}$ и предсказания $\hat{\mathbf{y}}$, $\Omega$ – функция штрафа за норму параметров $\mathbf{w}$, $\lambda$ — малое значение, которое контролирует силу регуляризации. Двумя наиболее часто используемыми функциями штрафа за норму параметра являются $\mathcal{L}_1(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_{j=1}^{d} |w_j|$ и $\mathcal{L}_2(\mathbf{w}) = \|\mathbf{w}\|_2 = \sum_{j=1}^{d} w_j^2$.

Параметры глубоких нейронных сетей часто имеют абсолютные значения меньше 1, поэтому функция $\mathcal{L}_1$ может привести к большему штрафу, чем $\mathcal{L}_2$, поскольку $|x| > x^2$ при $|x| < 1$. Следовательно, функция потерь со штрафом $\mathcal{L}_1$ в большей степени побуждает параметры нейронной сети иметь достаточно малые значения или даже нули.

# Параметры при регуляризациях $L_1$ и $L_2$




Изображение контурных линий исходной функции потерь (красный) и функций штрафа (синий). Точки пересечения контурных линий указывают на то, что штрафная функция $\mathcal{L}_1$ может приводить к параметрам с нулевым значением.

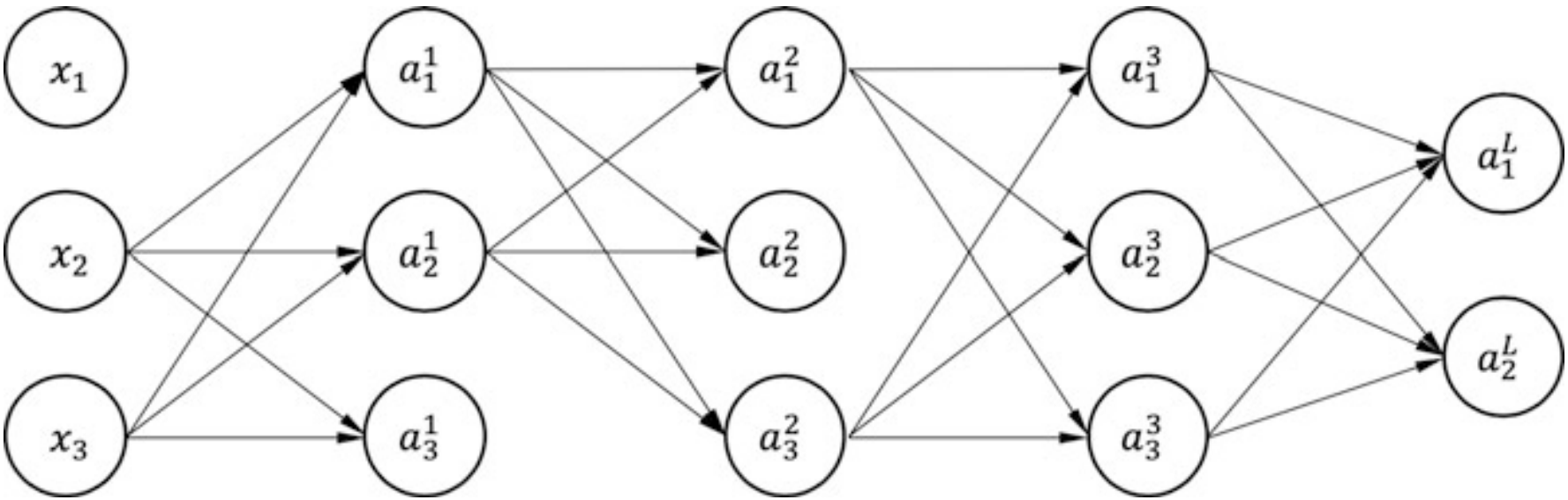
# Отбор признаков при регуляризации $L_1$

Использование функции потерь с штрафом $\mathcal{L}_1$ позволяет нейронной сети неявно выполнять отбор признаков, то есть отбрасывать некоторые входные признаки, устанавливая соответствующие параметры нейронной сети равными нулю или некоторым небольшим значениям. Как было показано на предыдущем слйде, при заданных параметрах $w_1$, $w_2$ в системе координат множество точек $w_1^2 + w_2^2 = r^2$ представляет собой окружность с радиусом $r$, а множество точек $|w_1| + |w_2| = r$ – квадрат с длиной диагонали $2\,r$, оба множества которых показаны синими контурными линиями. Красные контурные линии соответствуют исходной функции потерь $\mathcal{L}\left(\mathbf{y}, \hat{\mathbf{y}}\right)$. Точки пересечения штрафов за норму параметров и первоначальных убытков, обозначенные красными крестиками, указывают на то, что функция штрафа $\mathcal{L}_1$ с большей вероятностью приведет к параметрам с нулевым значением, чем функция штрафа $\mathcal{L}_2$, в то время как функция штрафа $\mathcal{L}_2$ может привести к параметрам с близкими абсолютными значениями, но отличными от нулевых.

# Исключение нейронов (дропаут)

Глубокие нейронные сети с большим количеством нейронов могут страдать от взаимной адаптации нейронов, что может привести к переобучению. Взаимная адаптация нейронов означает, что нейроны зависят друг от друга.

Дропаут (dropout) представляет собой метод предотвращения совместной адаптации нейронов, когда при обучении нейронной сети выходы скрытых слоев случайным образом обнуляются, что напоминает случайное отсоединение нейронов.



# Эффект от применения дропаута

Термин «dropout» (отсев, выбивание, отбрасывание) характеризует исключение определённого процента (например 30%) случайных нейронов (находящихся как в скрытых, так и видимых слоях) на разных итерациях (эпохах) во время обучения нейронной сети.

Во время обратного распространения ошибки при нулевом выходе соответствующая частная производная функции потерь по отношению к выходу слоя будет равна нулю. Поэтому веса «отключенных» нейронов обновляться не будут, а будут обновляться только оставшиеся подключенными нейроны.

Поэтому, метод дропаута может обучать разные подсети нейронной сети, позволяя всем им использовать одни и те же параметры. Это очень эффективный способ усреднения моделей внутри нейронной сети. В результате более обученные нейроны получают в сети больший вес. Во время прогнозирования (тестирования) дропаут отключается, и никакие выходы слоев не обнуляются. Это означает, что все подсети работают вместе, чтобы предсказать конечный результат.

Дропаут значительно увеличивает скорость обучения, качество обучения на тренировочных данных, а также повышает качество предсказаний модели на новых тестовых данных.

# Пакетная нормализация

Пакетная нормализация (batch-normalization) – это метод, который позволяет повысить производительность и стабилизировать работу искусственных нейронных сетей. Суть данного метода заключается в том, что некоторым слоям нейронной сети на вход подаются данные, предварительно обработанные и имеющие нулевое математическое ожидание и единичную дисперсию.

Во время обучения слой пакетной нормализации оценивает среднее значение и дисперсию входных данных в пакете, используя скользящее среднее. Далее скользящие среднее значение и дисперсия обновляются, чтобы нормализовать входные данные пакета. Во время тестирования (прогнозирования) среднее значение и дисперсия фиксированы и применяются для нормализации входных данных.

Помимо повышения производительности и стабильности, пакетная нормализация обеспечивает регуляризацию. Подобно процессу исключения, который добавляет случайный фактор к скрытым значениям, скользящее среднее значение и дисперсия нормализации пакета вносят случайность, поскольку они обновляются на каждой итерации в соответствии со случайным мини-пакетом.

# Свойства пакетной нормализации

Кроме того, использование пакетной нормализации обладает еще несколькими дополнительными полезными свойствами:

- достигается более быстрая сходимость моделей, несмотря на выполнение дополнительных вычислений;
- пакетная нормализация позволяет каждому слою сети обучаться более независимо от других слоев;
- становится возможным использование более высокого темпа обучения, так как пакетная нормализация гарантирует, что выходы узлов нейронной сети не будут иметь слишком больших или малых значений;
- пакетная нормализация в каком-то смысле также является механизмом регуляризации: данный метод привносит в выходы узлов скрытых слоев некоторый шум, аналогично методу dropout;
- модели становятся менее чувствительны к начальной инициализации весов.