

Лекция 2. Математические основы нейронных сетей



Дисциплина: **Интеллектуальный
данных, текстов и изображений**

анализ

Лектор: к.т.н. **Буров Сергей
Александрович**

burov-sa@ranepa.ru

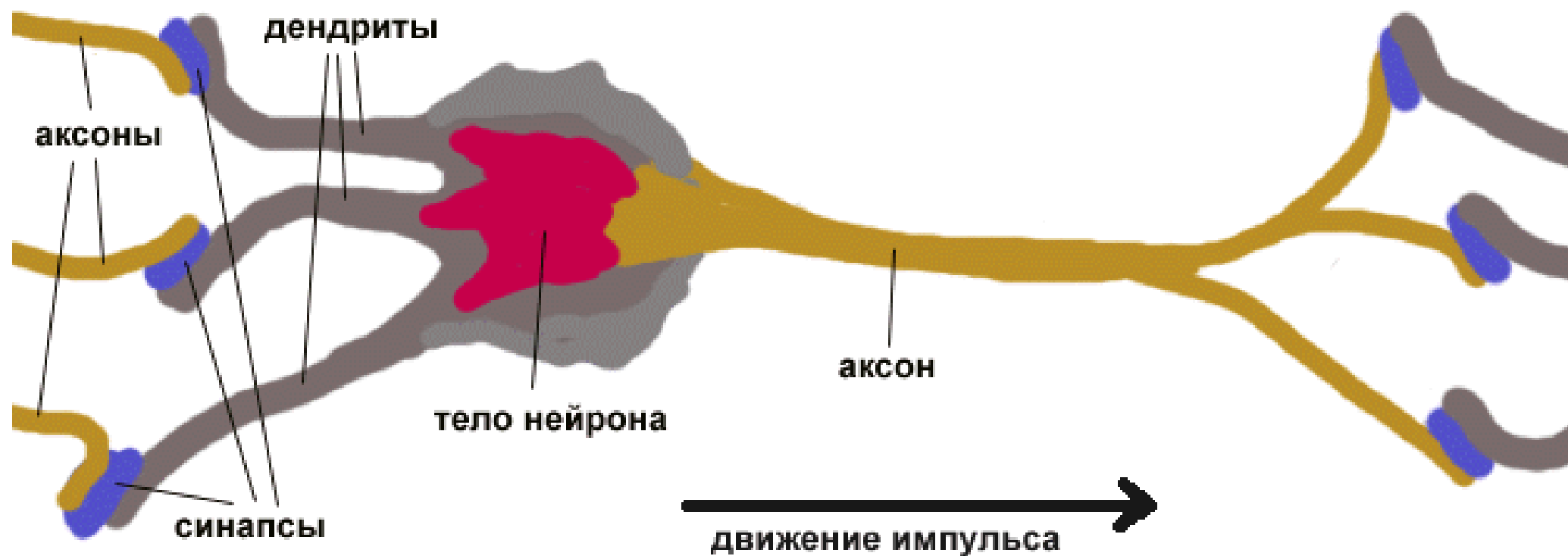
04.03.2024

Вопросы лекции:

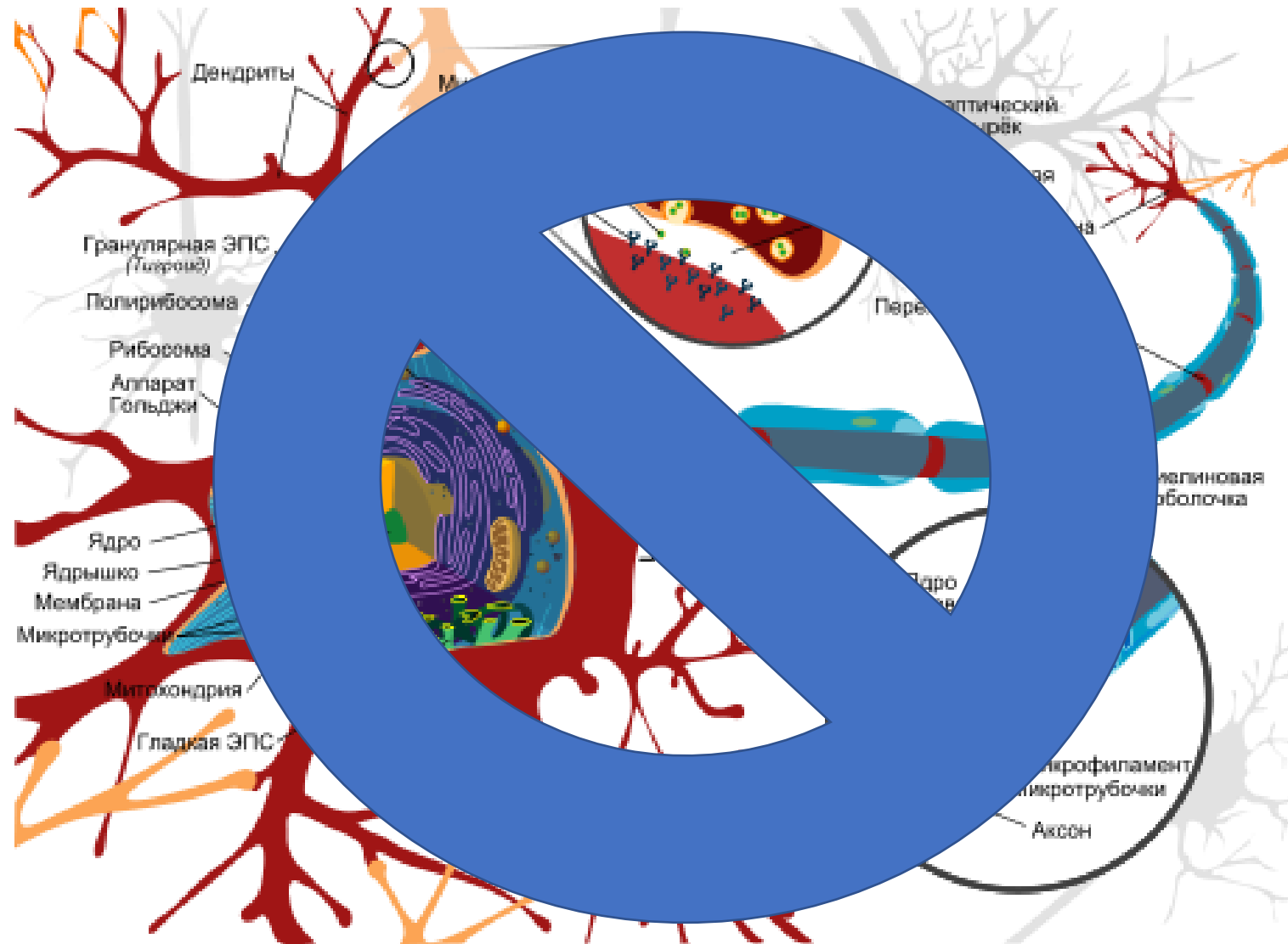
2

1. Основы обучения искусственных нейронных сетей. Методы градиентного спуска
2. Понятие тензора в машинном обучении
3. Основы работы с библиотекой NumPy

Введение



Введение

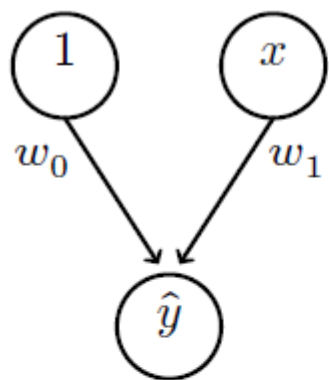


Нейросеть – вычислительный граф

От регрессии к нейросети

Линейная регрессия

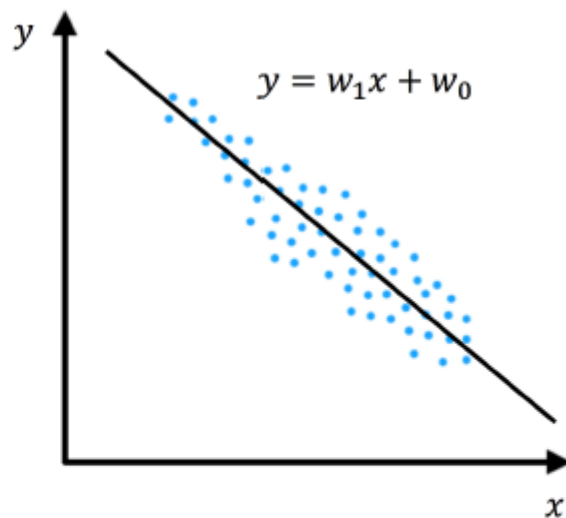
7



$$y_i = w_0 + w_1 \cdot x_i$$

$$y_i = \begin{pmatrix} 1 & x_i \end{pmatrix} \cdot \begin{pmatrix} w_0 \\ w_1 \end{pmatrix}$$

$$y_i = (x_i, w)$$



$$y_1 = w_0 + w_1 \cdot x_1$$

$$y_2 = w_0 + w_1 \cdot x_2$$

$$y_3 = w_0 + w_1 \cdot x_3$$

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \end{pmatrix} \cdot \begin{pmatrix} w_0 \\ w_1 \end{pmatrix}$$

Линейная регрессия (векторная форма)

8

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} \quad X = \begin{pmatrix} 1 & x_{11} & \dots & x_{1k} \\ 1 & x_{21} & \dots & x_{2k} \\ \dots & \dots & \dots & \dots \\ 1 & x_{n1} & \dots & x_{nk} \end{pmatrix} \quad w = \begin{pmatrix} w_0 \\ w_1 \\ \dots \\ w_k \end{pmatrix}$$

Модель:

$$y = Xw$$

Оценка:

$$\hat{w} = (X^T X)^{-1} X^T y$$

Прогноз:

$$\hat{y} = X\hat{w}$$

Как можно обучить линейную регрессию?

Например, можно ввести штраф за ошибку – **функцию потерь**

MSE (Mean Squared Error) – среднеквадратическая ошибка

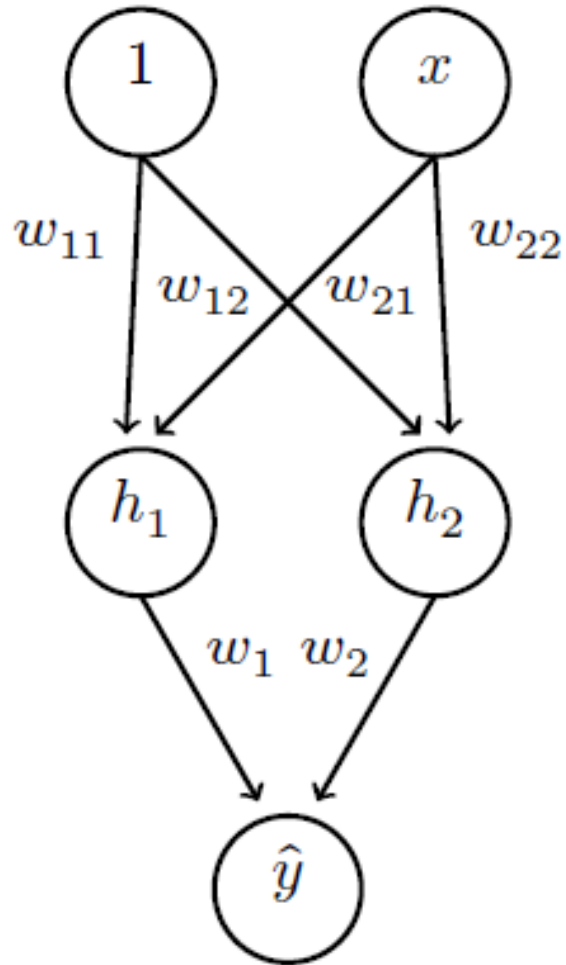
$$MSE(w) = \frac{1}{n} \sum_{i=1}^n (w^T x_i - y_i)^2 \rightarrow \min_w$$

MAE (Mean Absolute Error) – средняя абсолютная ошибка

$$MAE(w) = \frac{1}{n} \sum_{i=1}^n |w^T x_i - y_i| \quad \text{- не всегда подходит}$$

Усложним этот вычислительный граф

10



$$h_{1i} = w_{11} + w_{21} \cdot x_i$$

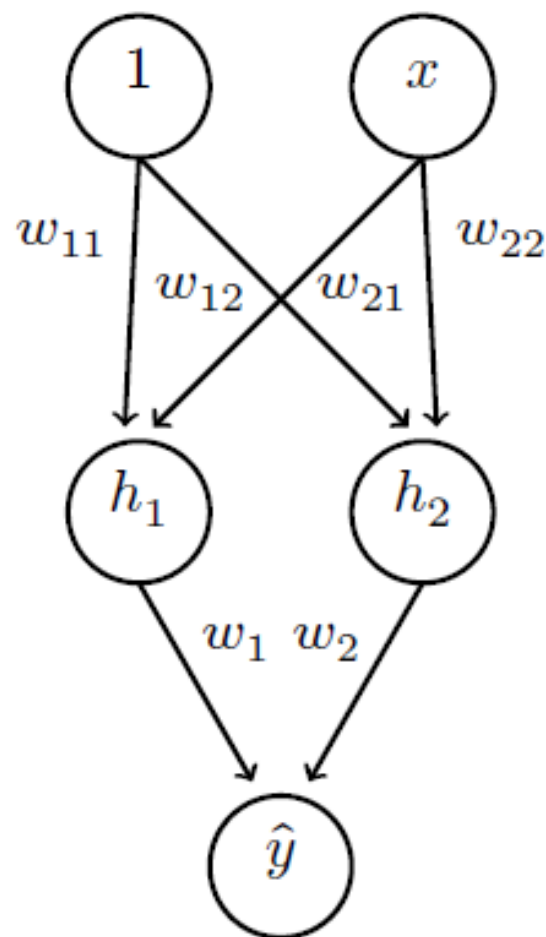
$$h_{2i} = w_{12} + w_{22} \cdot x_i$$

$$y_i = w_1 \cdot h_{1i} + w_2 \cdot h_{2i}$$

$$h = X \cdot W_1$$

$$y = h \cdot W_2$$

Добавим нелинейность



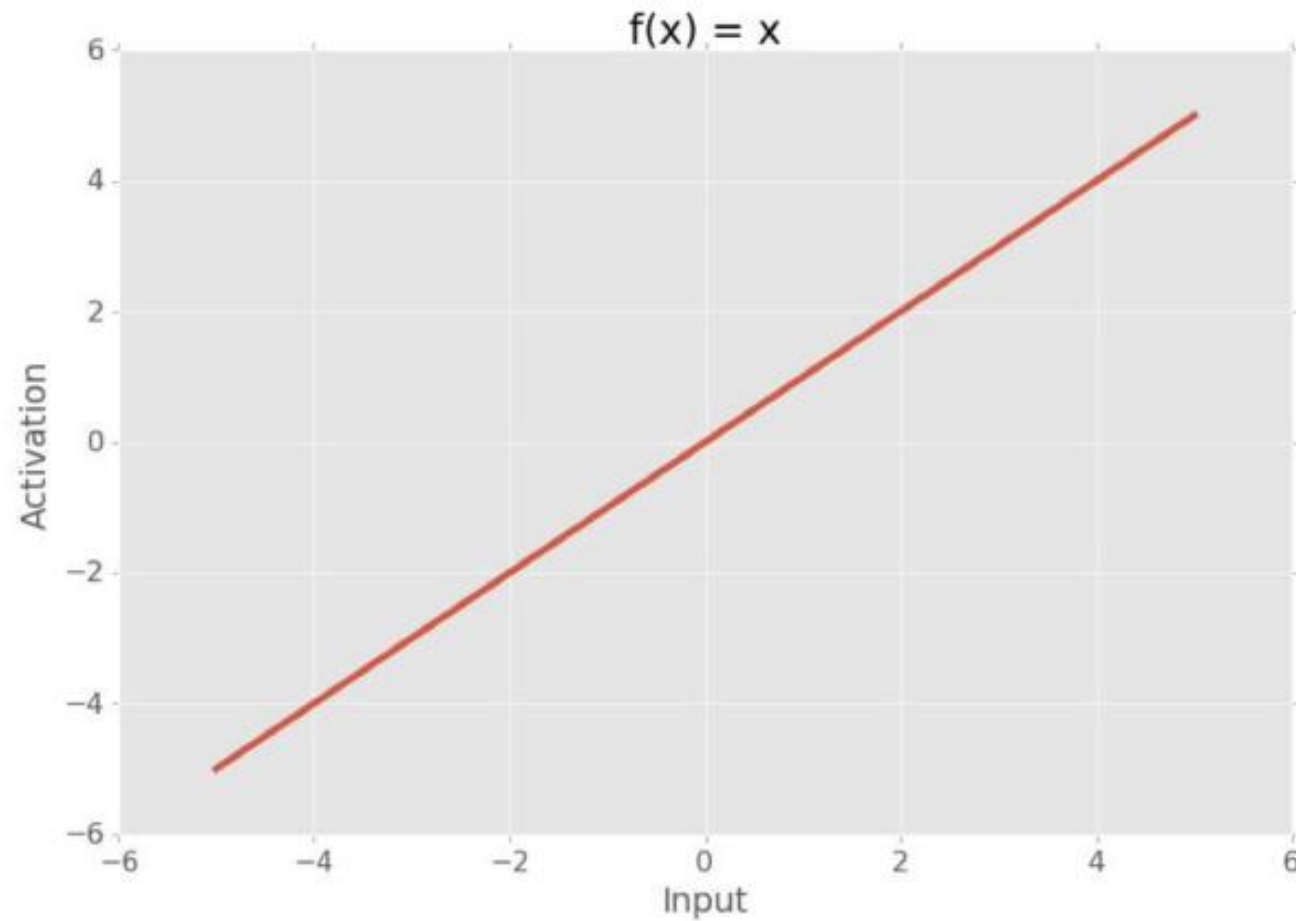
$$h_{1i} = w_{11} + w_{21} \cdot x_i$$

$$h_{2i} = w_{12} + w_{22} \cdot x_i$$

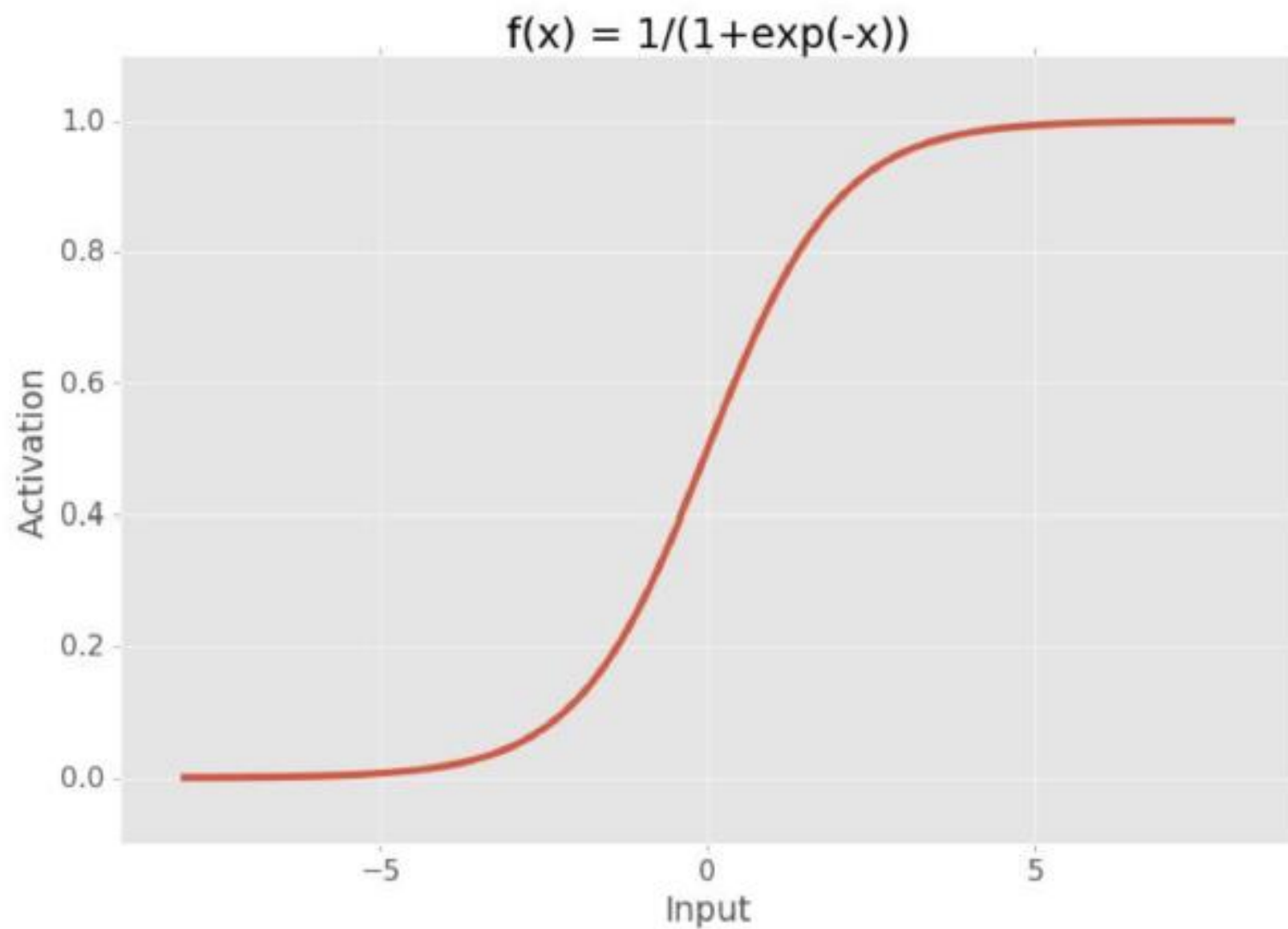
$$y_i = w_1 \cdot f(h_{1i}) + w_2 \cdot f(h_{2i})$$

$$y = h \cdot W_2 = f(X \cdot W_1) \cdot W_2 \neq X \cdot A$$

Тождественная функция активации

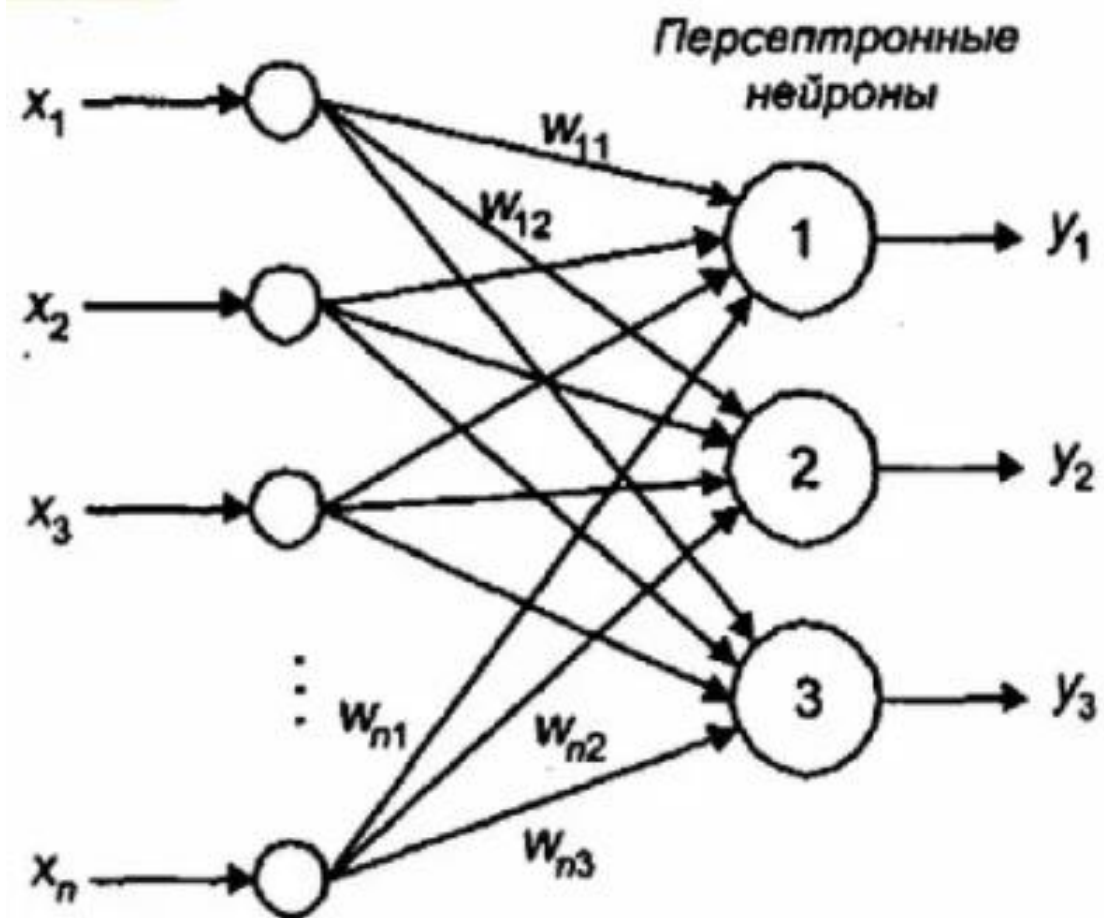


Сигмоида



Перцептрон

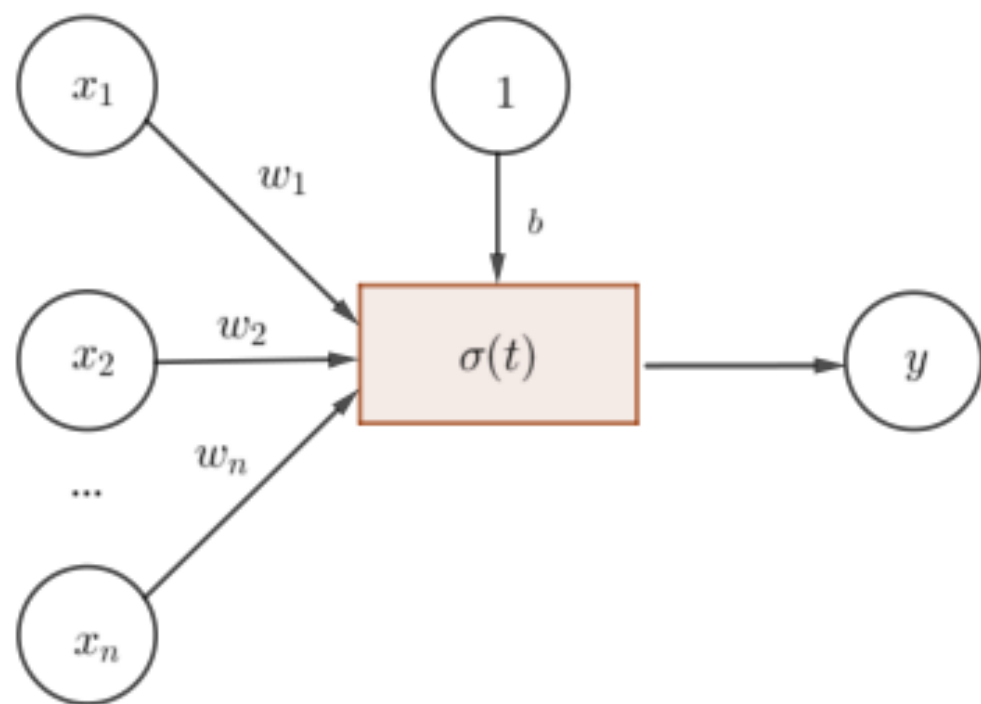
14



$$y_j = f\left(\sum_{i=1}^n x_i w_{ij}\right), j = 1..3$$

$$f(x) = \frac{1}{1 + e^{-\alpha x}}$$

$$f'(x) = \alpha \cdot f(x) \cdot (1 - f(x))$$



Нейрон с сигмоидом в качестве функции активации — это логистическая регрессия...

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

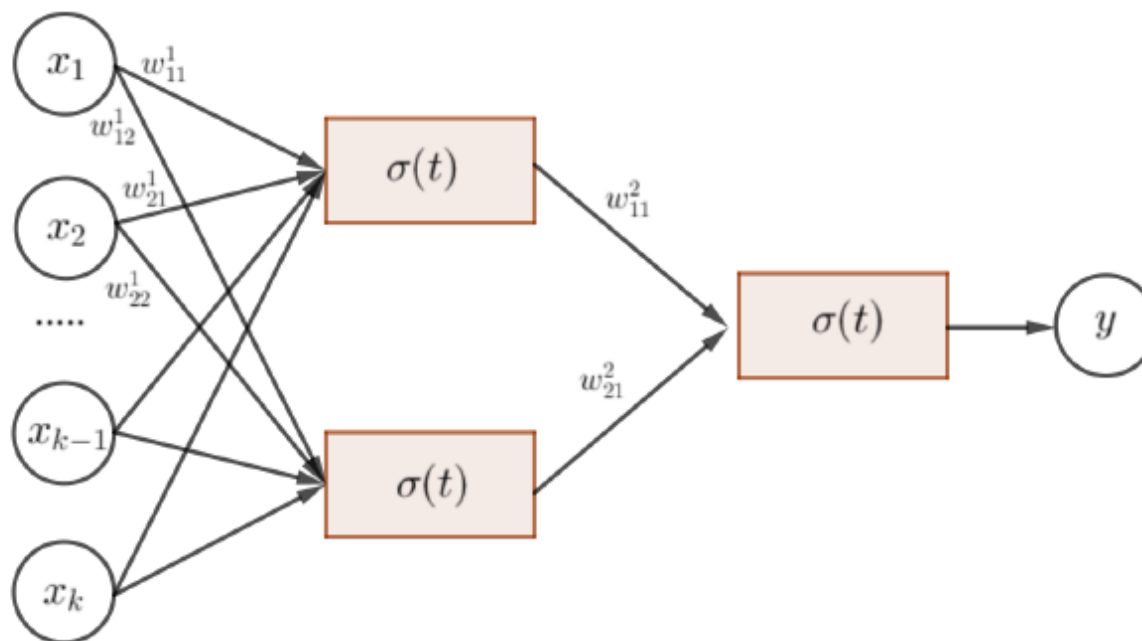
$$P(y = 1 \mid x) = \sigma(w_0 + w_1 \cdot x_1 + \dots + w_n \cdot x_n)$$

Что такое нейросеть?

16

Нейросеть это **последовательная комбинация регрессий**

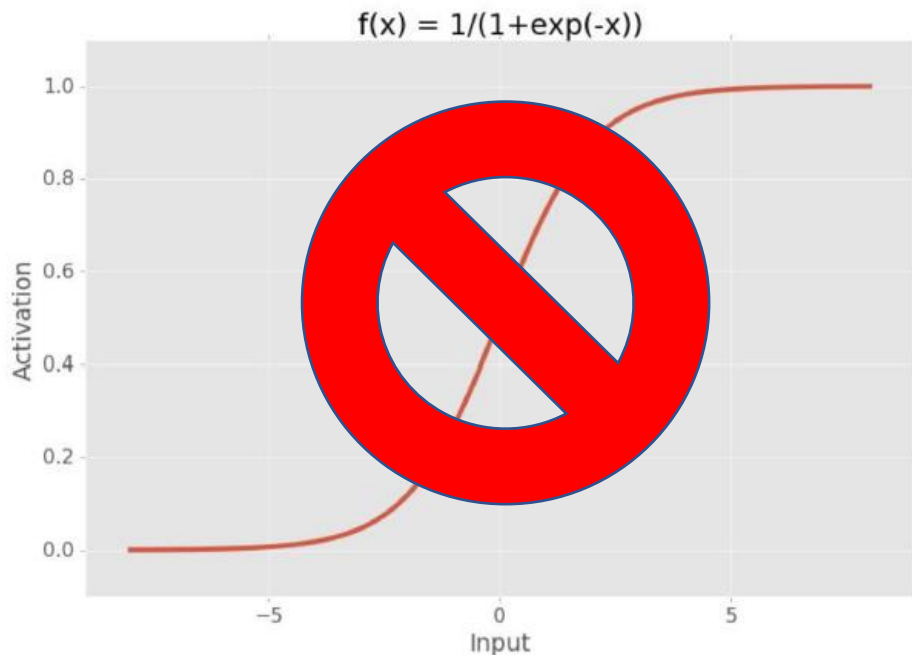
Две регрессии скреплены третьей



$$h_j = \sigma(w_0 + w_{j1}^1 \cdot x_1 + \dots + w_{jk}^1 \cdot x_k)$$

$$y = \sigma(w_{11}^2 \cdot h_1 + w_{21}^2 \cdot h_2)$$

Можно ли обойтись перцептроном и сигмойдой?




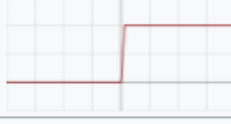


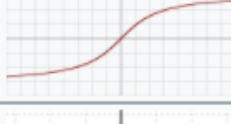


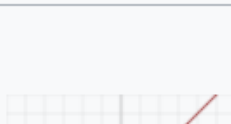
The Perceptron Convergence Theorem (Rosenblat, 1965)

1. Любая непрерывная и ограниченная функция может быть сколь угодно точно аппроксимирована нейронной сетью с одним скрытым слоем с нелинейной функцией активации нейрона.
2. Любая функция может быть сколь угодно точно аппроксимирована нейронной сетью с двумя скрытыми слоями с нелинейной функцией активации нейрона.

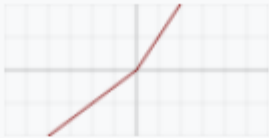
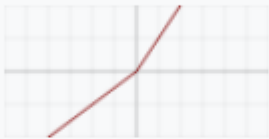


Сигмоида хорошо работает в небольших нейросетях, но при большом количестве нейронов, например, в глубоких сетях, возникает **паралич сети**

Функции активации

18

Название	График	Уравнение	Производная (по x)
Тождественная		$f(x) = x$	$f'(x) = 1$
Единичная ступенька		$f(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & x \neq 0 \\ ? & x = 0 \end{cases}$
Логистическая (сигмоида или Гладкая ступенька)		$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$ ^[1]	$f'(x) = f(x)(1 - f(x))$
th		$f(x) = \text{th}(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$f'(x) = 1 - f(x)^2$
arctg		$f(x) = \text{tg}^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Softsign ^{[9][10]}		$f(x) = \frac{x}{1 + x }$	$f'(x) = \frac{1}{(1 + x)^2}$
Обратный квадратный корень (англ. <i>Inverse square root unit</i> , ISRU) ^[11]		$f(x) = \frac{x}{\sqrt{1 + \alpha x^2}}$	$f'(x) = \left(\frac{1}{\sqrt{1 + \alpha x^2}} \right)^3$
Линейный выпрямитель ^[en] (или Полулинейный элемент) (англ. <i>Rectified linear unit</i> , ReLU) ^{[12][13]}		$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$

Функции активации

<p>Линейный выпрямитель с «утечкой» (англ. <i>Leaky rectified linear unit</i>, Leaky ReLU)^[14]</p>		$f(x) = \begin{cases} 0,01x & x < 0 \\ x & x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0,01 & x < 0 \\ 1 & x \geq 0 \end{cases}$
<p>Параметрический линейный выпрямитель (англ. <i>Parameteric rectified linear unit</i>, PReLU)^[15]</p>		$f(\alpha, x) = \begin{cases} \alpha x & x < 0 \\ x & x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & x < 0 \\ 1 & x \geq 0 \end{cases}$
<p>Рандомизированный линейный выпрямитель с «утечкой» (англ. <i>Randomized leaky rectified linear unit</i>, RReLU)^[16]</p>		$f(\alpha, x) = \begin{cases} \alpha x & x < 0 \\ x & x \geq 0 \end{cases} \text{ [3]}$	$f'(\alpha, x) = \begin{cases} \alpha & x < 0 \\ 1 & x \geq 0 \end{cases}$
<p>Экспоненциальная линейная функция (англ. <i>Exponential linear unit</i>, ELU)^[17]</p>		$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & x < 0 \\ x & x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} f(\alpha, x) + \alpha & x < 0 \\ 1 & x \geq 0 \end{cases}$

Желательные свойства функции активации

Нелинейность – Если функция активации нелинейна, можно доказать, что двухуровневая нейронная сеть будет универсальным аппроксиматором функции, Тожественная функция активации не удовлетворяет этому свойству. Если несколько уровней используют тождественную функцию активации, вся сеть эквивалентна одноуровневой модели.

Непрерывная дифференцируемость – Это свойство желательно (RELU не является непрерывно дифференцируемой и имеет некоторые проблемы с оптимизацией, основанной на градиентном спуске, но остаётся допустимой возможностью) для обеспечения методов оптимизации на основе градиентного спуска. Двоичная ступенчатая функция активации не дифференцируема в точке 0 и её производная равна 0 во всех других точках, так что методы градиентного спуска не дают никакого успеха для неё

Область значений – Если множество значений функции активации ограничено, методы обучения на основе градиента более стабильны, поскольку представления эталонов существенно влияют лишь на ограниченный набор весов связей. Если область значений бесконечна, обучение, как правило, более эффективно, поскольку представления эталонов существенно влияют на большинство весов. В последнем случае обычно необходим меньший темп обучения.

Монотонность – Если функция активации монотонна, поверхность ошибок, ассоциированная с одноуровневой моделью, гарантированно будет выпуклой.

Процесс обучения нейросети

Прямое распространение ошибки (forward propagation):

$$X \Rightarrow X \cdot W_1 \Rightarrow f(X \cdot W_1) \Rightarrow f(X \cdot W_1) \cdot W_2 \Rightarrow \dots \Rightarrow \hat{y}$$

Считаем потери:

$$Loss = \frac{1}{2}(y - \hat{y})^2$$

Для обучения нужно использовать градиентный спуск

Как обучить нейросеть?

$$L(W_1, W_2) = \frac{1}{2} \cdot (y - f(X \cdot W_1) \cdot W_2)^2$$

Секрет успеха в умении брать производную и градиентном спуске.

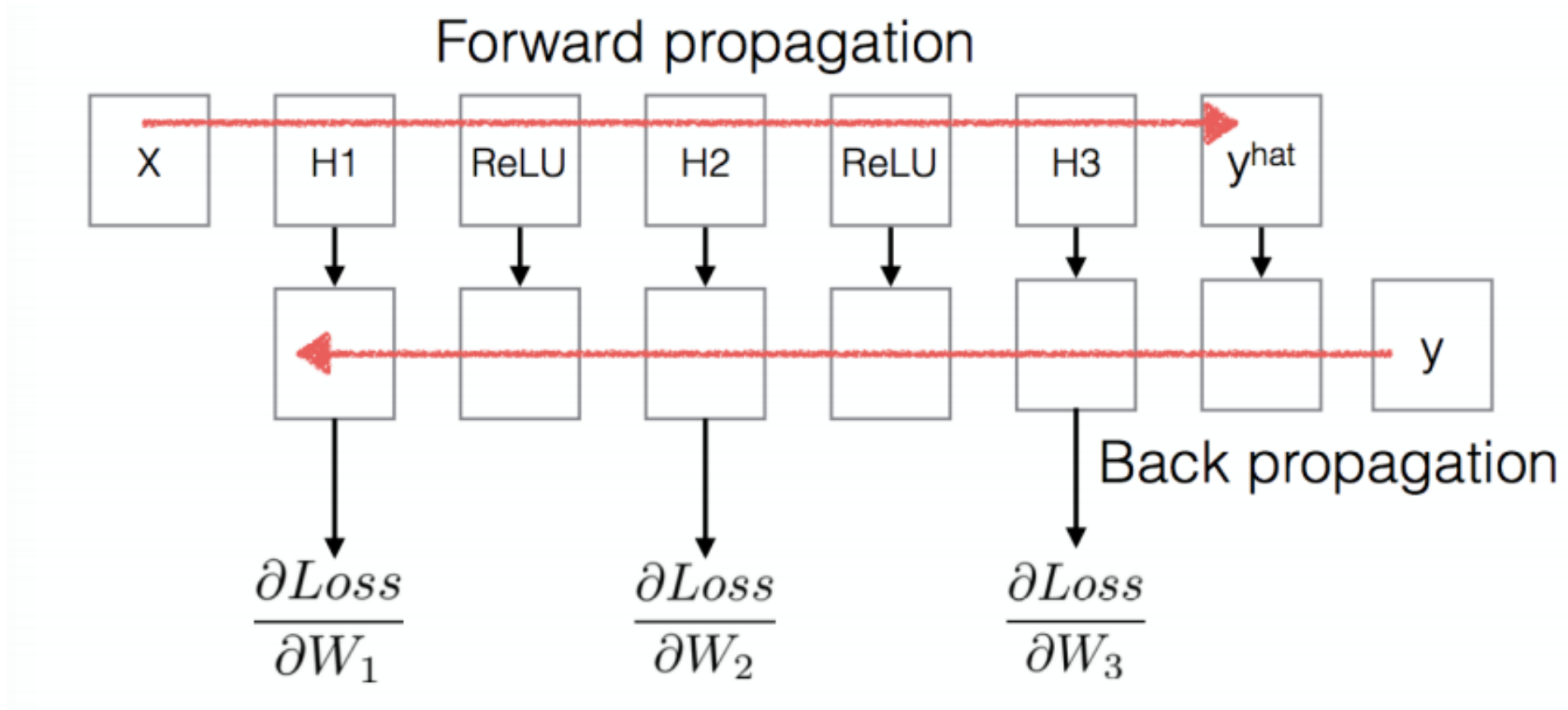
$$f(g(x))' = f'(g(x)) \cdot g'(x)$$

$$\frac{\partial L}{\partial W_2} = -(y - f(X \cdot W_1) \cdot W_2) \cdot f(X \cdot W_1)$$

$$\frac{\partial L}{\partial W_1} = -(y - f(X \cdot W_1) \cdot W_2) \cdot W_2 f'(X \cdot W_1) \cdot W_1$$

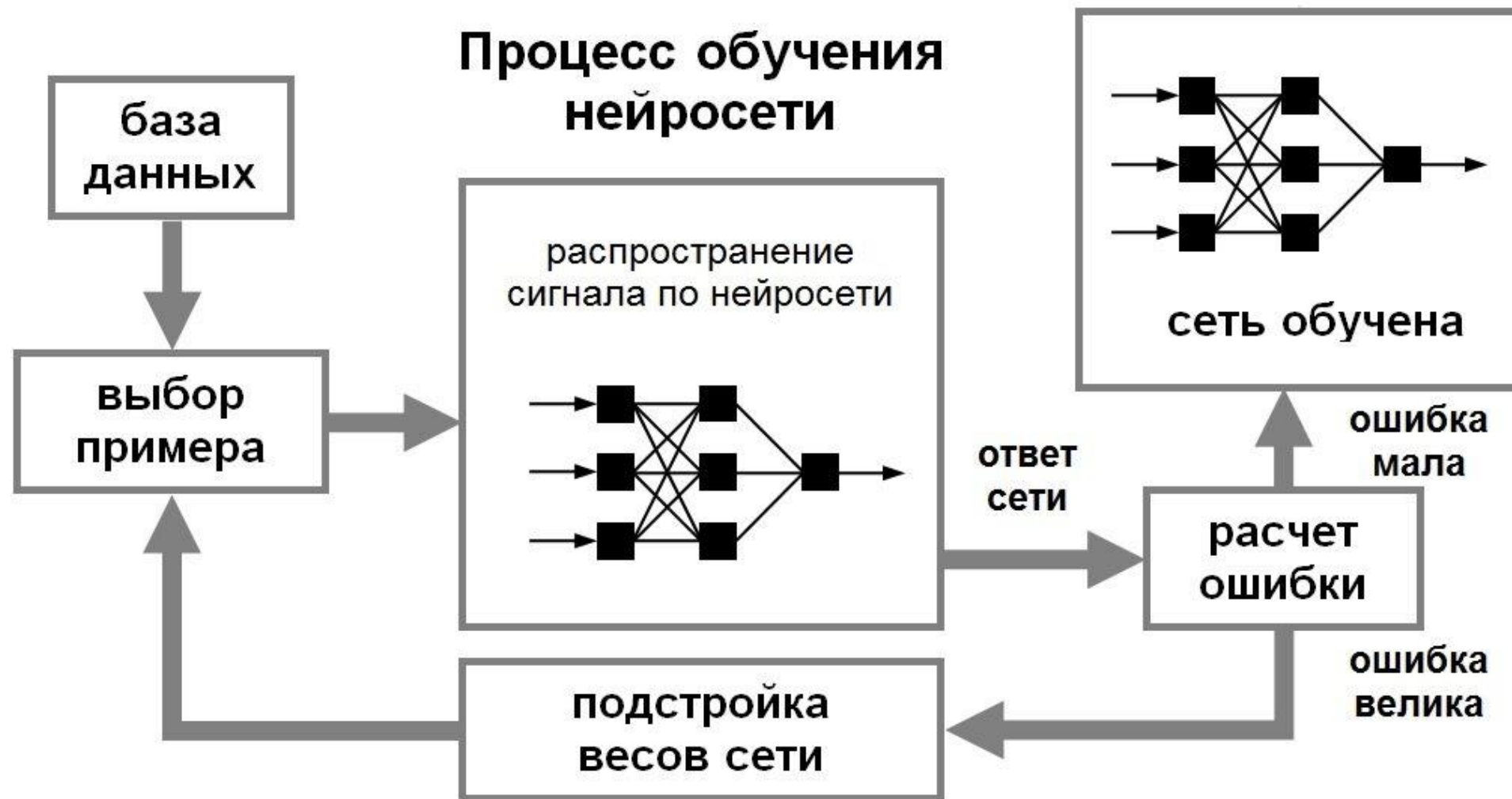
Дважды ищем одно и то же \Rightarrow оптимизация поиска производных даст нам алгоритм обратного распространения ошибки (back-propagation)

Алгоритм об работного распространения ошибки



Процесс обучения нейросети

24



Градиент

25

Градиент - это вектор, касающийся функции и указывающий в направлении наибольшего увеличения этой функции. Градиент равен нулю при локальном максимуме или минимуме, потому что нет единого направления увеличения. В математике градиент определяется как частная производная для каждой входной переменной функции. Например, у нас есть функция:

$$f(x, y) = \frac{1}{2}(x^2 + y^2)$$

мы видим, что минимум функции равен (0, 0).

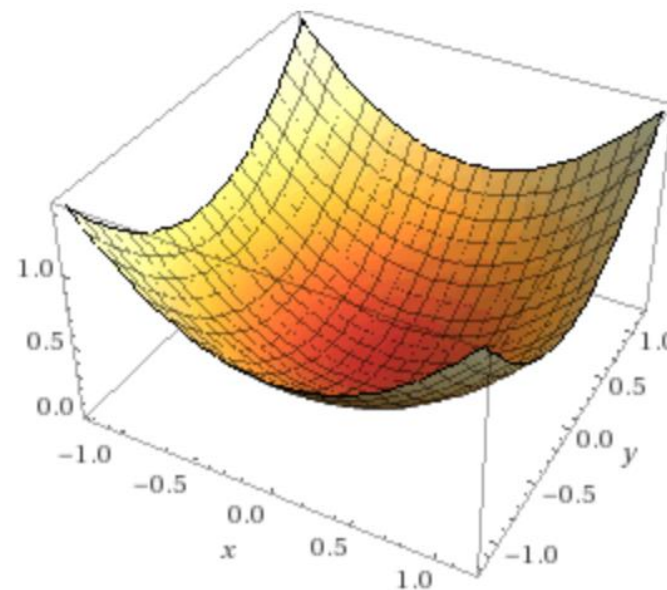
В этом случае x-компонент градиента является частной производной по x, а y-компонент градиента является частной производной по y. Градиент функции выше:

$$\nabla f(x, y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) = (x, y)$$

Если мы хотим найти направление, которое увеличивает функцию максимум в точке (1, 2), мы можем подключить (1, 2) к формуле выше и получить:

$$\textit{Direction} = (1, 2)$$

3D plot:



Градиентный спуск

Поскольку градиент - это вектор, указывающий на наибольшее увеличение функции, отрицательный градиент - это вектор, указывающий на наибольшее уменьшение функции. Следовательно, мы можем минимизировать функцию, итеративно немного двигаясь в направлении отрицательного градиента. Это логика градиентного спуска.

Учитывая отправную точку $(x_1^{(0)}, \dots, x_n^{(0)})$

Мы можем построить итерационную процедуру:

$$\begin{aligned}x_1^{(i+1)} &= x_1^{(i)} - a \cdot \frac{\partial f}{\partial x_1}(x^{(i)}) \\x_n^{(i+1)} &= x_n^{(i)} - a \cdot \frac{\partial f}{\partial x_n}(x^{(i)})\end{aligned}$$

Параметр **альфа** в приведенной выше формуле называется **скоростью обучения**, которая в большинстве случаев является небольшой константой и находится в диапазоне от 0 до 1. Итерационная процедура не остановится, пока не сойдется. Взяв предыдущий пример, мы уже знали, что градиент:

Градиентный спуск

Поэтому итерационная процедура градиентного спуска может быть записана как:

$$x^{(i+1)} = x^{(i)} - a \cdot \frac{\partial f}{\partial x}(x^{(i)}, y^{(i)}) = (1 - a) \cdot x^{(i)}$$

$$y^{(i+1)} = y^{(i)} - a \cdot \frac{\partial f}{\partial y}(x^{(i)}, y^{(i)}) = (1 - a) \cdot y^{(i)}$$

Тогда мы можем получить:

$$x^{(i+1)} = (1 - a)^i \cdot x^{(0)}$$

$$y^{(i+1)} = (1 - a)^i \cdot y^{(0)}$$

Наконец, предполагая, что альфа меньше 1, тогда мы можем получить:

$$\lim(x^{(i)}, y^{(i)}) = (0, 0)$$

$$\nabla f(x, y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) = (x, y)$$

Градиентный спуск

Поэтому итерационная процедура градиентного спуска может быть записана как:

$$x^{(i+1)} = x^{(i)} - a \cdot \frac{\partial f}{\partial x}(x^{(i)}, y^{(i)}) = (1 - a) \cdot x^{(i)}$$

$$y^{(i+1)} = y^{(i)} - a \cdot \frac{\partial f}{\partial y}(x^{(i)}, y^{(i)}) = (1 - a) \cdot y^{(i)}$$

Тогда мы можем получить:

$$x^{(i+1)} = (1 - a)^i \cdot x^{(0)}$$

$$y^{(i+1)} = (1 - a)^i \cdot y^{(0)}$$

Наконец, предполагая, что альфа меньше 1, тогда мы можем получить:

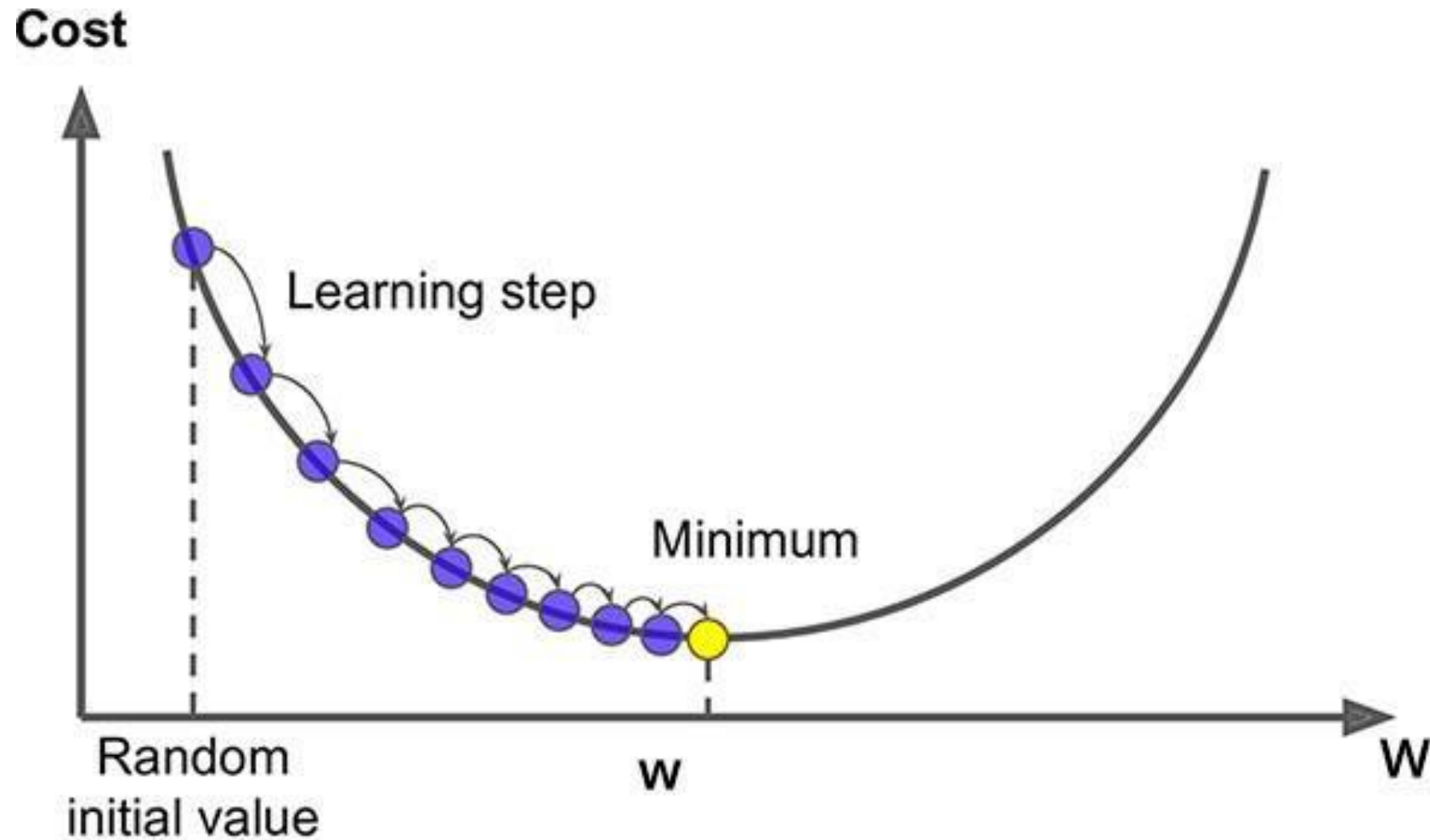
$$\lim(x^{(i)}, y^{(i)}) = (0, 0)$$

$$\nabla f(x, y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) = (x, y)$$

Градиентный спуск

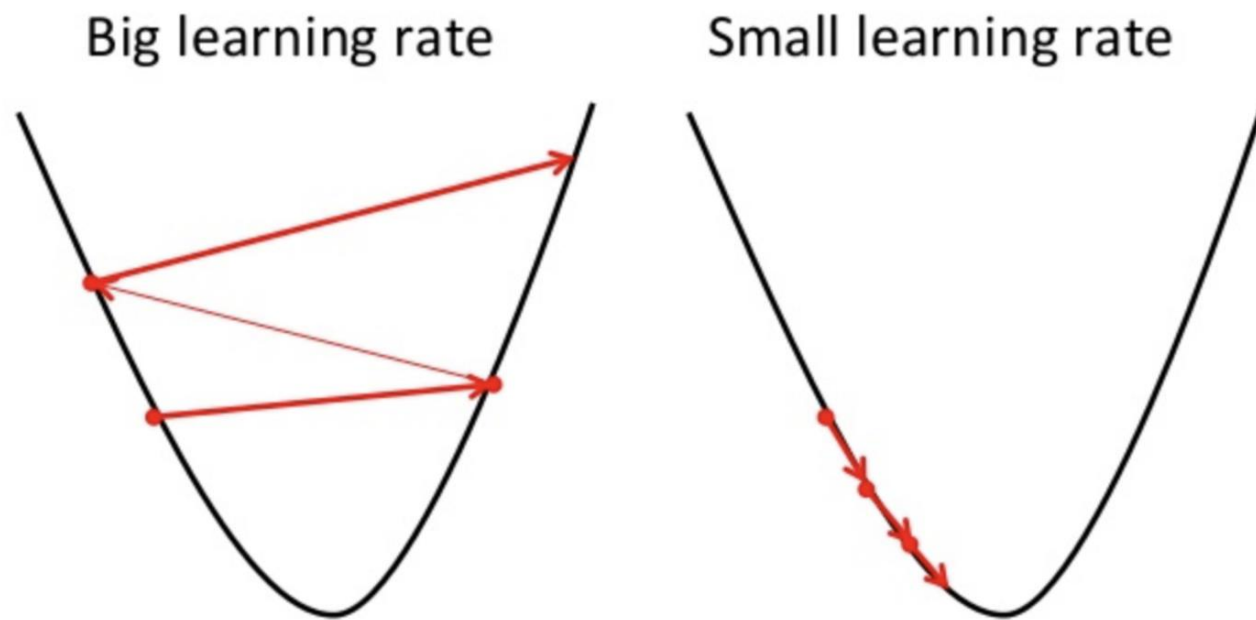
29

Градиентный спуск – метод итеративной оптимизации для поиска минимума функции



Градиентный спуск. Влияние скорости обучения

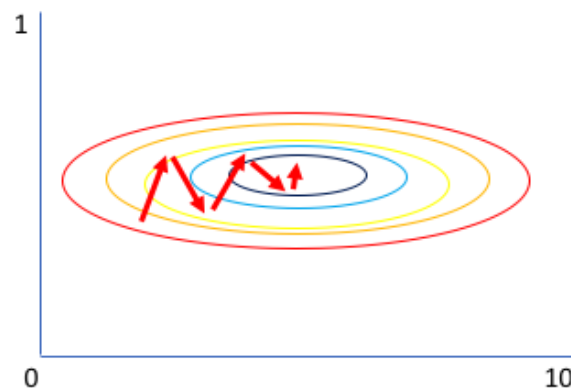
Когда скорость обучения слишком велика, градиентный спуск может перепрыгнуть через долину и оказаться на другой стороне. Это приведет к расхождению функции стоимости. С другой стороны, когда скорость обучения слишком мала, алгоритм будет сходиться долго. Следовательно, перед началом градиентного спуска необходима правильная скорость обучения.



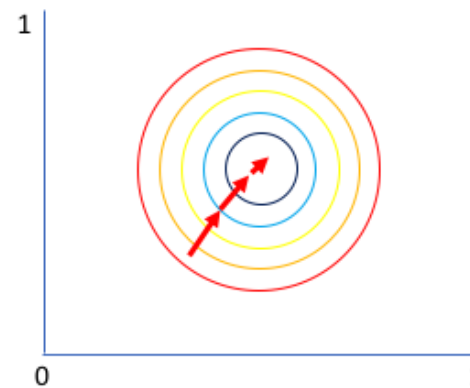
Градиентный спуск. Нормализация

Нормализация играет важную роль для градиентного спуска. Если функции не нормализованы, в обновлении будут преобладать объекты с большим масштабом, поэтому алгоритм будет генерировать зигзагообразный путь обучения. Требуется много ненужных шагов и больше времени, чтобы достичь минимума. После нормализации всех функций функция стоимости приобретает более сферическую форму. Алгоритм градиентного спуска идет прямо к минимуму. Один из способов нормализации - это минус среднее и деление на стандартное отклонение.

Why normalize?



Gradient of larger parameter
dominates the update



Both parameters can be
updated in equal proportions

Виды градиентного спуска

Пакетный градиентный спуск

Пакетный градиентный спуск использует всю партию обучающих данных на каждом шаге. Он рассчитывает ошибку для каждой записи и принимает среднее значение для определения градиента. Преимущество Batch Gradient Descent заключается в том, что алгоритм более эффективен в вычислительном отношении и обеспечивает стабильный путь обучения, что облегчает сходимость. Тем не менее, пакетный градиентный спуск занимает больше времени, когда тренировочный набор большой.

Стохастический градиентный спуск

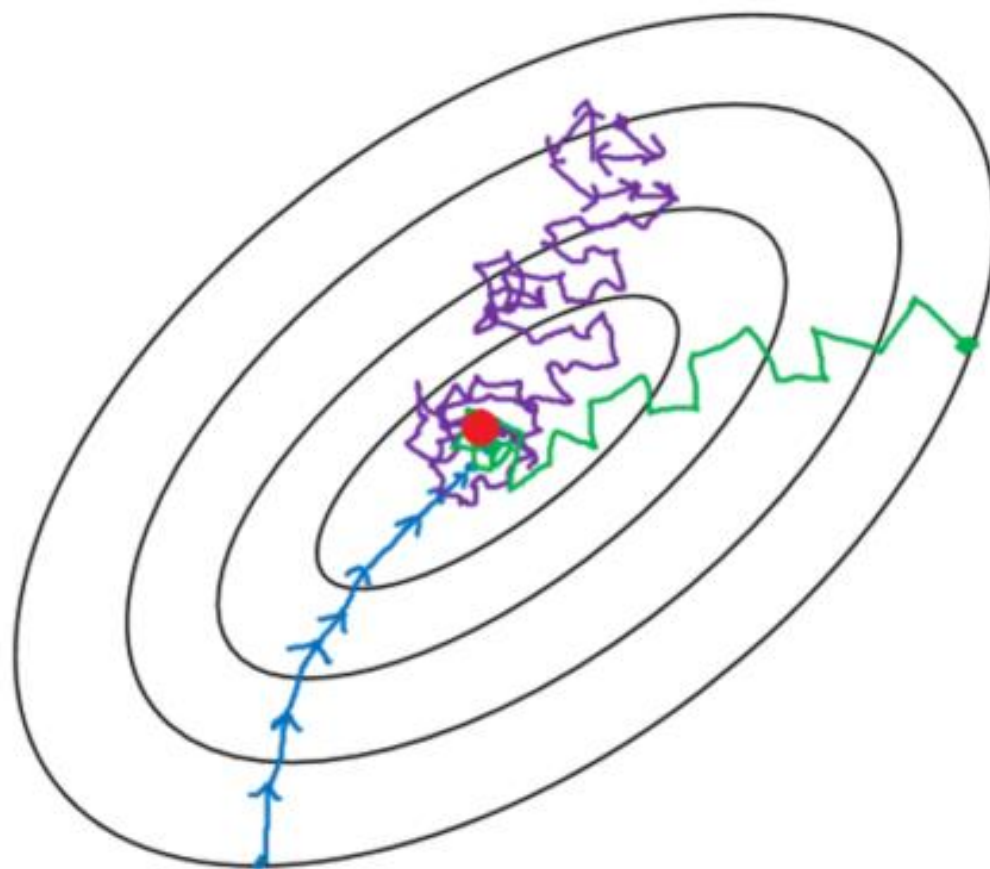
В другом крайнем случае Stochastic Gradient Descent просто выбирает один экземпляр из обучающего набора на каждом шаге и обновляет градиент только на основе этой отдельной записи. Преимущество Stochastic Gradient Descent заключается в том, что алгоритм намного быстрее на каждой итерации, что устраняет ограничение в Batch Gradient Descent. Однако алгоритм дает менее регулярный и стабильный путь обучения по сравнению с пакетным градиентным спуском. Вместо плавного уменьшения функция стоимости будет подпрыгивать вверх и вниз. После раундов итераций алгоритм может найти хороший параметр, но конечный результат не обязательно является глобальным оптимальным.

Мини-пакетный градиентный спуск

Мини-пакетный градиентный спуск сочетает в себе концепцию пакетного и стохастического градиентного спуска. На каждом этапе алгоритм вычисляет градиент на основе подмножества обучающего набора вместо полного набора данных или только одной записи. Преимущество мини-пакетного градиентного спуска в том, что алгоритм может использовать преимущества матричной операции во время расчета, а функция стоимости может уменьшаться более плавно и стабильно, чем стохастический градиентный спуск.

Виды градиентного спуска

33



- Batch gradient descent
- Mini-batch gradient Descent
- Stochastic gradient descent

Пакетный градиентный спуск (GD)

Проблема оптимизации:

$$L(w) = \frac{1}{n} \cdot \sum_{i=1}^n L(w, x_i, y_i) \rightarrow \min_w$$

Градиент указывает направление максимального роста

$$\nabla L(w) = \left(\frac{\partial L(w)}{\partial w_0}, \frac{\partial L(w)}{\partial w_2}, \dots, \frac{\partial L(w)}{\partial w_k} \right)$$

Пакетный градиентный спуск

Проблема оптимизации:

$$L(w) = \frac{1}{n} \cdot \sum_{i=1}^n L(w, x_i, y_i) \rightarrow \min_w$$

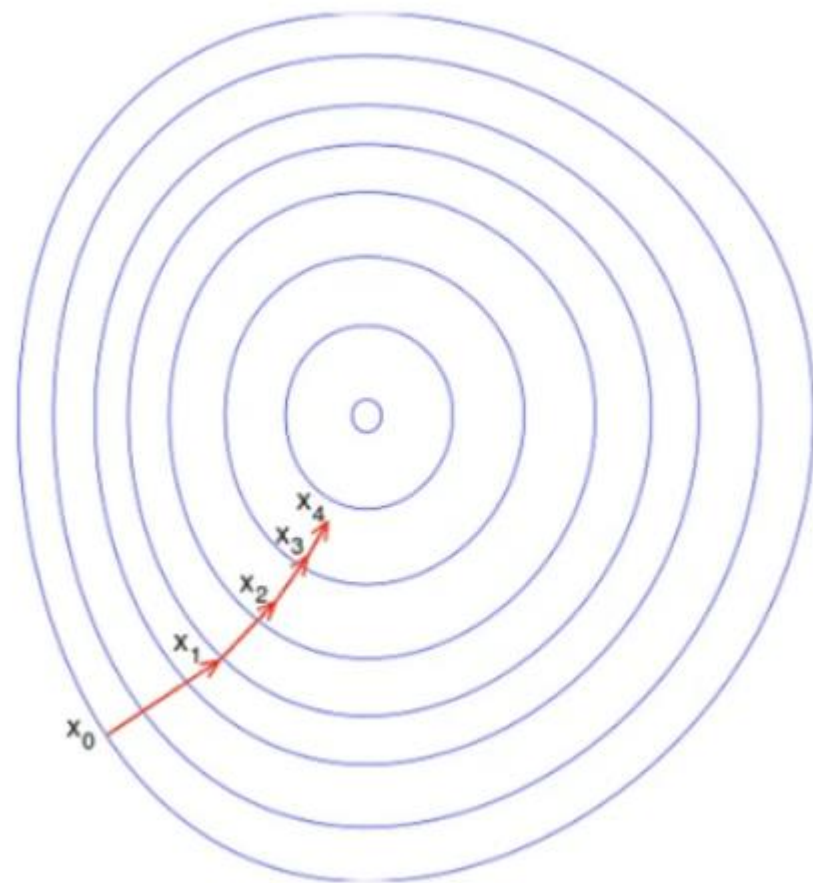
Градиент указывает направление максимального роста

$$\nabla L(w) = \left(\frac{\partial L(w)}{\partial w_0}, \frac{\partial L(w)}{\partial w_1}, \dots, \frac{\partial L(w)}{\partial w_k} \right)$$

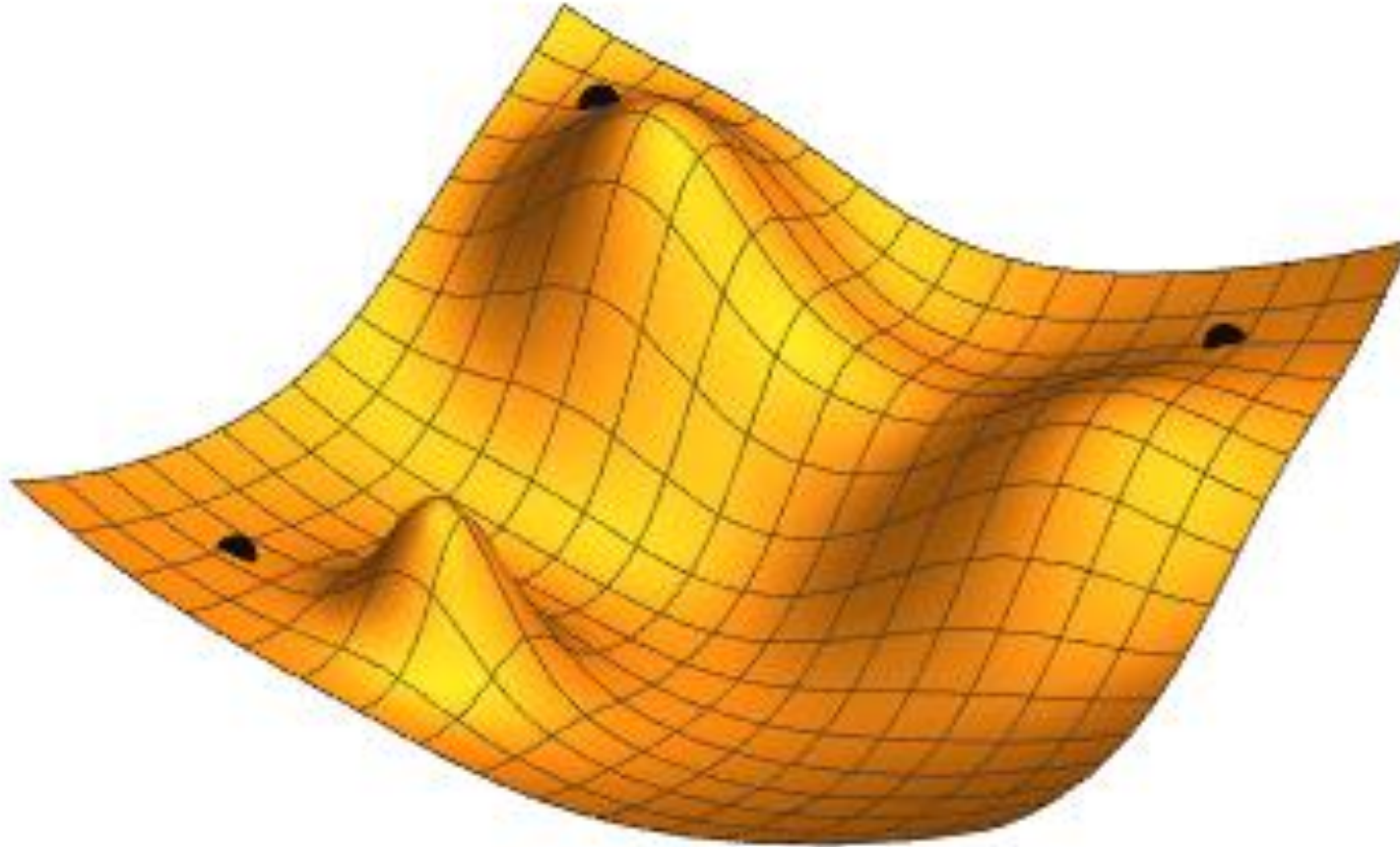
Идём в противоположную сторону:

$$w^1 = w^0 - \underset{\text{скорость обучения}}{\eta} \cdot \nabla L(w^0)$$

Градиентный спуск



Градиентный спуск



Градиентный спуск

38

Пример:

Проблема оптимизации:

$$L(w) = \frac{1}{n} \cdot \sum_{i=1}^n (y_i - x_i^T w)^2 \rightarrow \min_w$$

Градиент:

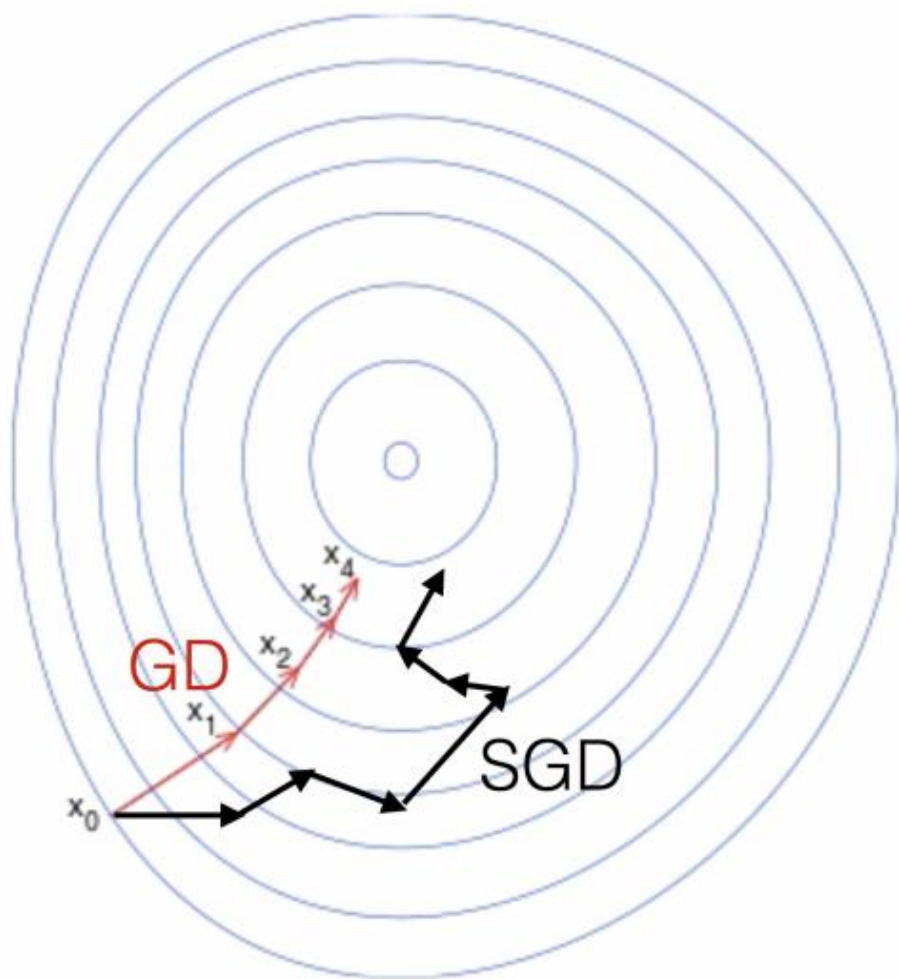
$$\nabla L(w) = -2 \cdot \frac{1}{n} \cdot \sum_{i=1}^n (y_i - x_i^T w) \cdot x_i$$

Идём в противоположную сторону:

$$w^1 = w^0 + 0.001 \cdot 2 \cdot \frac{1}{n} \cdot \sum_{i=1}^n (y_i - x_i^T w) \cdot x_i$$

Дорого постоянно считать такие суммы!

Стохастический градиентный спуск (**SGD**)



- И для GD и для SGD нет гарантий глобального минимума, сходимости
- SGD быстрее, на каждой итерации используется только одно наблюдение
- Для SGD спуск очень зашумлён
- GD: $O(n)$, SGD: $O(1)$

Мини пакетный градиентный спуск (Mini-bathSGD)

Проблема оптимизации:

$$L(w) = \sum_{i=1}^n L(w, x_i, y_i) \rightarrow \min_w$$

Инициализация w_0

while True:

 рандомно выбрали $m < n$ индексов

$$g_t = \frac{1}{m} \sum_{i=1}^m \nabla L(w, x_i, y_i)$$

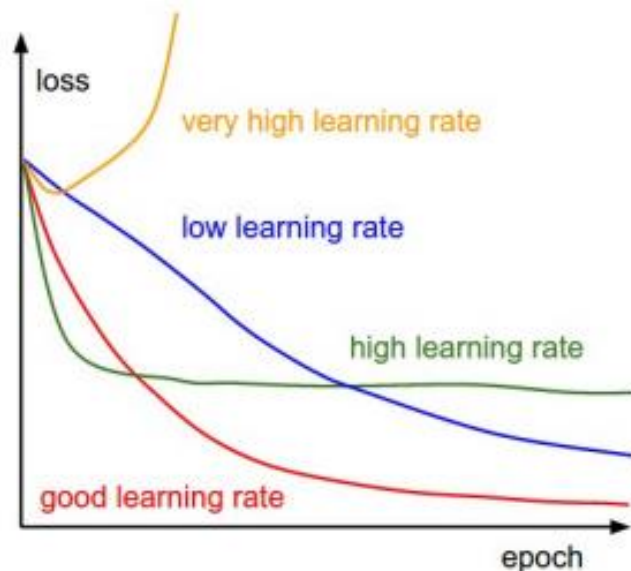
$$w_t = w_{t-1} - \eta_t \cdot g_t$$

if $\|w_t - w_{t-1}\| < \varepsilon$:

break

Проблемы при градиентном спуске

Скорость обучения η надо подбирать аккуратно, если она будет большой, мы можем скакать вокруг минимума, если маленькой - вечно ползти к нему.



К обновлению всех параметров применяется одна и та же скорость обучения. Возможно, что какие-то параметры приходят в оптимальную точку быстрее, и их не надо обновлять.

Метод моментов (Momentum SGD)

Momentum SGD

Мы считали на каждом шаге градиент по формуле

$$g_t = \frac{1}{m} \sum_{i=1}^m \nabla L(w_{t-1}, x_i, y_i).$$

После шага мы забывали его. **Давайте запоминать направление:**

$$h_t = \alpha \cdot h_{t-1} + \eta \cdot g_t$$

$$w_t = w_{t-1} - h_t$$

- Движение поддерживается в том же направлении, что и на предыдущем шаге
- Нет резких изменений направления движения.
- Обычно $\alpha = 0.9$.

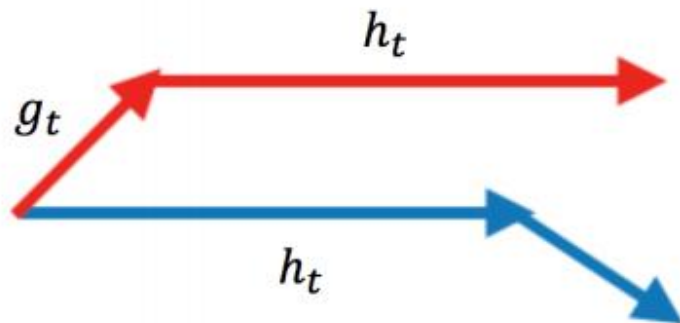
Метод моментов

(Momentum SGD)

- Бежим с горки и всё больше ускоряемся в том направлении, в котором были направлены сразу несколько предыдущих градиентов, но при этом движемся медленно там, где градиент постоянно меняется
- Хотелось бы не просто бежать с горы, но и хотя бы на полшага посмотреть себе под ноги, чтобы внезапно не споткнуться \Rightarrow давайте посмотреть на градиент в будущей точке
- Согласно методу моментов $\alpha \cdot h_{t-1}$ точно будет использоваться при шаге, давайте искать $\nabla L(w_{t-1} - \alpha \cdot h_{t-1})$.

(Nesterov Momentum SGD)

- Мы теперь сначала прыгаем в том же направлении, в каком шли до этого, потом корректируем его (голубая траектория).



$$h_t = \alpha \cdot h_{t-1} + \eta \cdot \nabla L(w_{t-1} - \alpha \cdot h_{t-1})$$

$$w_t = w_{t-1} - h_t$$

Адаптивные методы градиентного спуска

Разная скорость обучения

- Может сложиться, что некоторые веса уже близки к своим локальным минимумам, по этим координатам надо двигаться медленнее, а по другим быстрее \Rightarrow **адаптивные методы градиентного спуска**
- Шаг изменения должен быть меньше у тех параметров, которые в большей степени варьируются в данных, и больше у тех, которые менее изменчивы

Адаптивный градиентный спуск (AdaGrad)

$$G_t^j = G_{t-1}^j + g_{tj}^2$$
$$w_t^j = w_{t-1}^j - \frac{\eta}{\sqrt{G_t^j + \varepsilon}} \cdot g_t^j$$

g_t^j — градиент по j —ому параметру

своя скорость обучения для каждого параметра

обычно $\eta = 0.01$, т.к. параметр не очень важен

G_t^j всегда увеличивается, из-за этого обучение может рано останавливаться \Rightarrow RMSprop

Стохастический градиентный спуск (RMSprop)

$$G_t^j = \alpha \cdot G_{t-1}^j + (1 - \alpha) \cdot g_{tj}^2$$
$$w_t^j = w_{t-1}^j - \frac{\eta_t}{\sqrt{G_t^j + \varepsilon}} \cdot g_t^j$$

Обычно $\alpha = 0.9$

Скорость обучения адаптируется к последнему сделанному шагу, бесконтрольного роста G_t^j больше не происходит

RMSprop нигде не был опубликован, Хинтон просто привёл его в своей лекции, сказав, что это норм тема

Стохастический градиентный спуск

(Adam – Adaptive Moment Estimation)

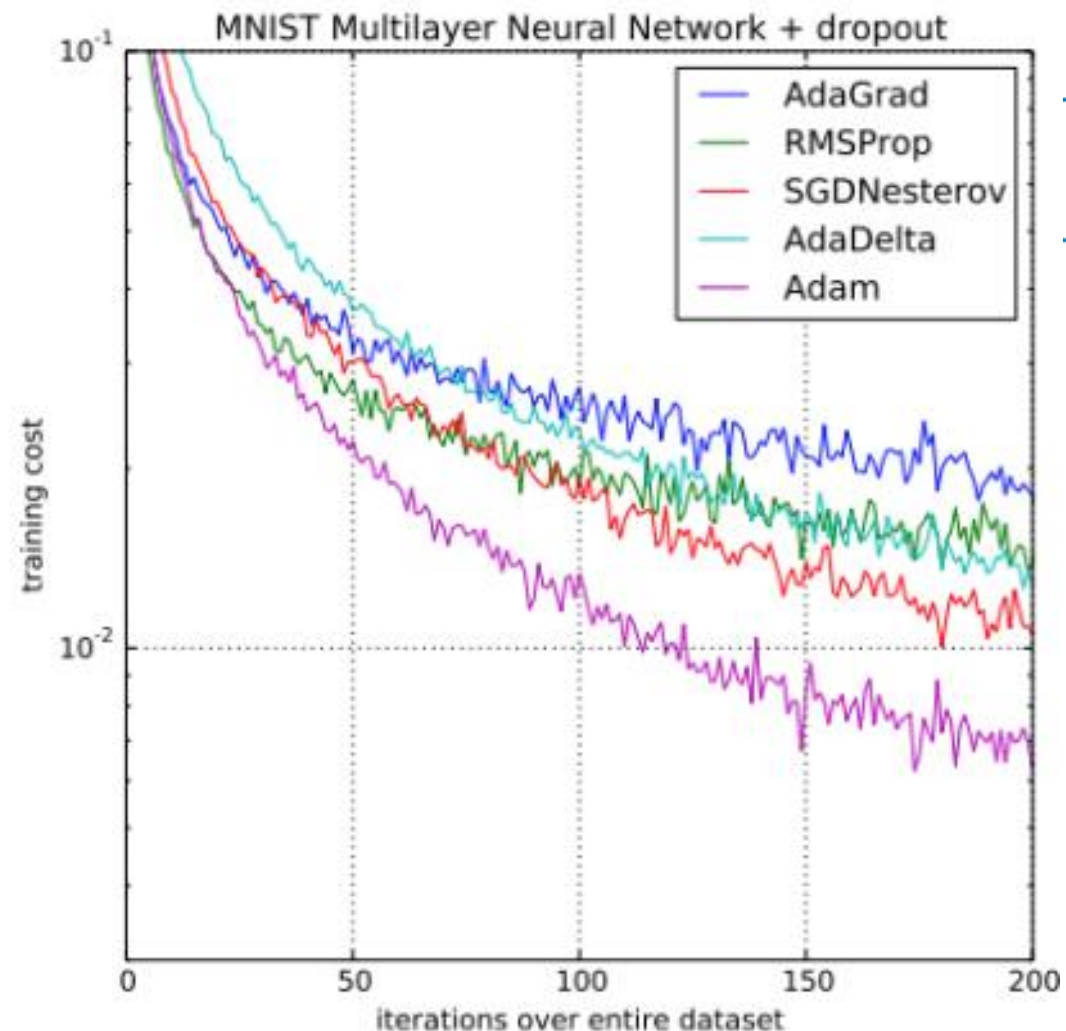
$$\begin{aligned}h_t^j &= \beta_1 \cdot h_{t-1}^j + (1 - \beta_1) \cdot g_{tj} \\G_t^j &= \beta_2 \cdot G_{t-1}^j + (1 - \beta_2) \cdot g_{tj}^2 \\w_t^j &= w_{t-1}^j - \frac{\eta_t}{\sqrt{G_t^j + \varepsilon}} \cdot h_t^j\end{aligned}$$

Комбинируем Momentum и индивидуальные скорости обучения

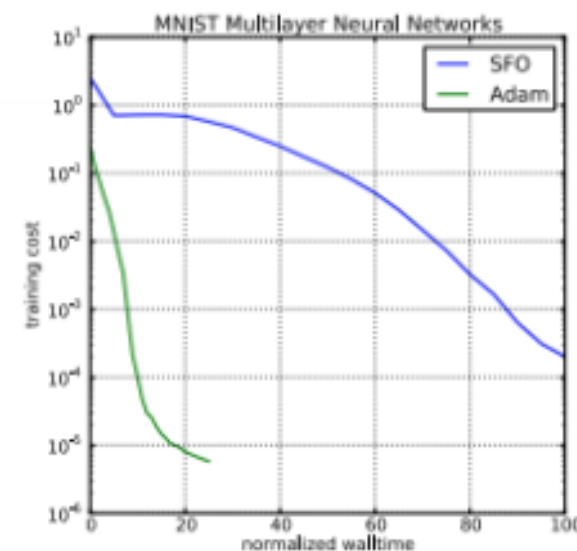
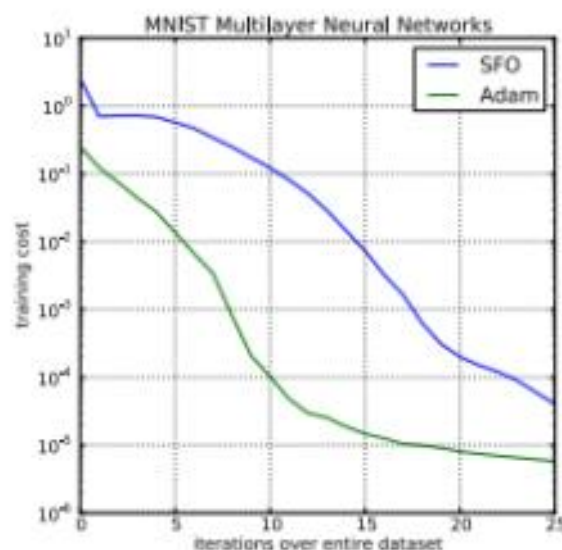
Фактически h_t и G_t это оценки первого и второго моментов для стохастического градиента

Сравнение методов градиентного спуска

49



- Momentum SGD сохраняет направление шага и позволяет добиваться более быстрой сходимости
- Адаптивные методы позволяют находить индивидуальную скорость обучения для каждого параметра
- Adam комбинирует в себе оба подхода



Понятие тензора в машинном обучении

50

Тензор 0-го уровня

(11)

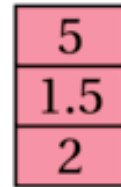
SCALAR

Тензор 1-го уровня



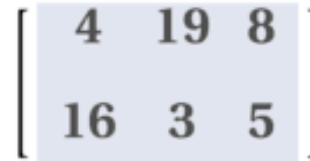
Row Vector
(shape 1x3)

array



Column Vector
(shape 3x1)

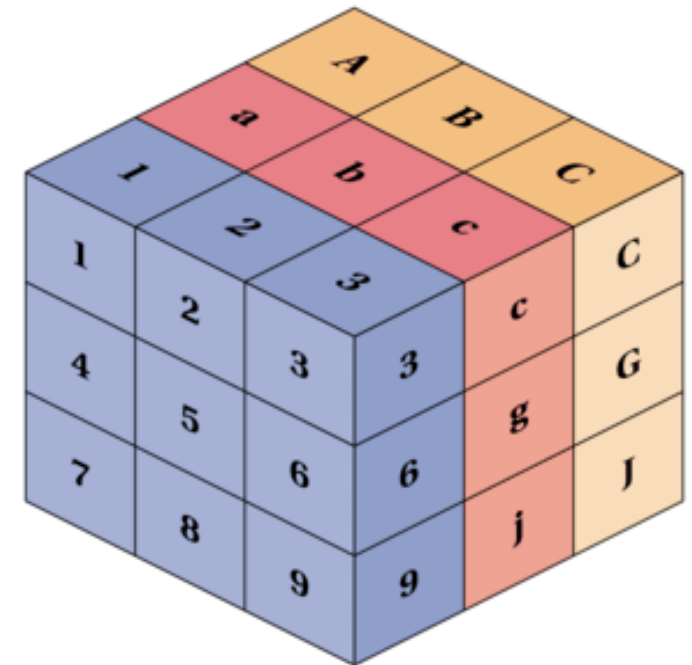
Тензор 2-го уровня



MATRIX

2d-array

Тензор 3-го уровня



TENSOR

nd-array
 $n = 3$

Свёртка

51

В математическом отношении в двумерной свертке нет ничего сложного. Имеется ядро – небольшая матрица весов. Это ядро «скользит» по двумерным входным данным, выполняя поэлементное умножение для той части данных, которую сейчас покрывает. Результаты перемножений ячеек суммируются в одном выходном пикселе. В случае сверточных нейросетей ядро определяется в ходе обучения сети. Начальные веса, аналогично случаю перцептрона, могут иметь случайные значения, и корректируются в процессе обучения

Про свёртку узнаем при изучении глубоких сетей

подробнее
изучении

3_0	3_1	2_2	1	0
0_2	0_2	1_0	3	1
3_0	1_1	2_2	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

<table border="1"><tr><td>0</td><td>-1</td><td>0</td></tr><tr><td>-1</td><td>5</td><td>-1</td></tr><tr><td>0</td><td>-1</td><td>0</td></tr></table>	0	-1	0	-1	5	-1	0	-1	0	*	<table border="1"><tr><td>5</td><td>6</td><td>7</td></tr><tr><td>3</td><td>4</td><td>6</td></tr><tr><td>2</td><td>3</td><td>5</td></tr></table>	5	6	7	3	4	6	2	3	5	=	$\sum \omega_i * x_i$	=	2
0	-1	0																						
-1	5	-1																						
0	-1	0																						
5	6	7																						
3	4	6																						
2	3	5																						
↑ Матрица свёртки ω_i		↑ Пиксели изображения x_i		↑ Взвешенная сумма: $0*5+(-1)*6+0*7+(-1)*3+5*4+(-1)*6+0*2+(-1)*3+0*5$		↑ Пиксель нового изображения																		
		↑ Операция свёртки																						

Понятие тензора

52

CAT?



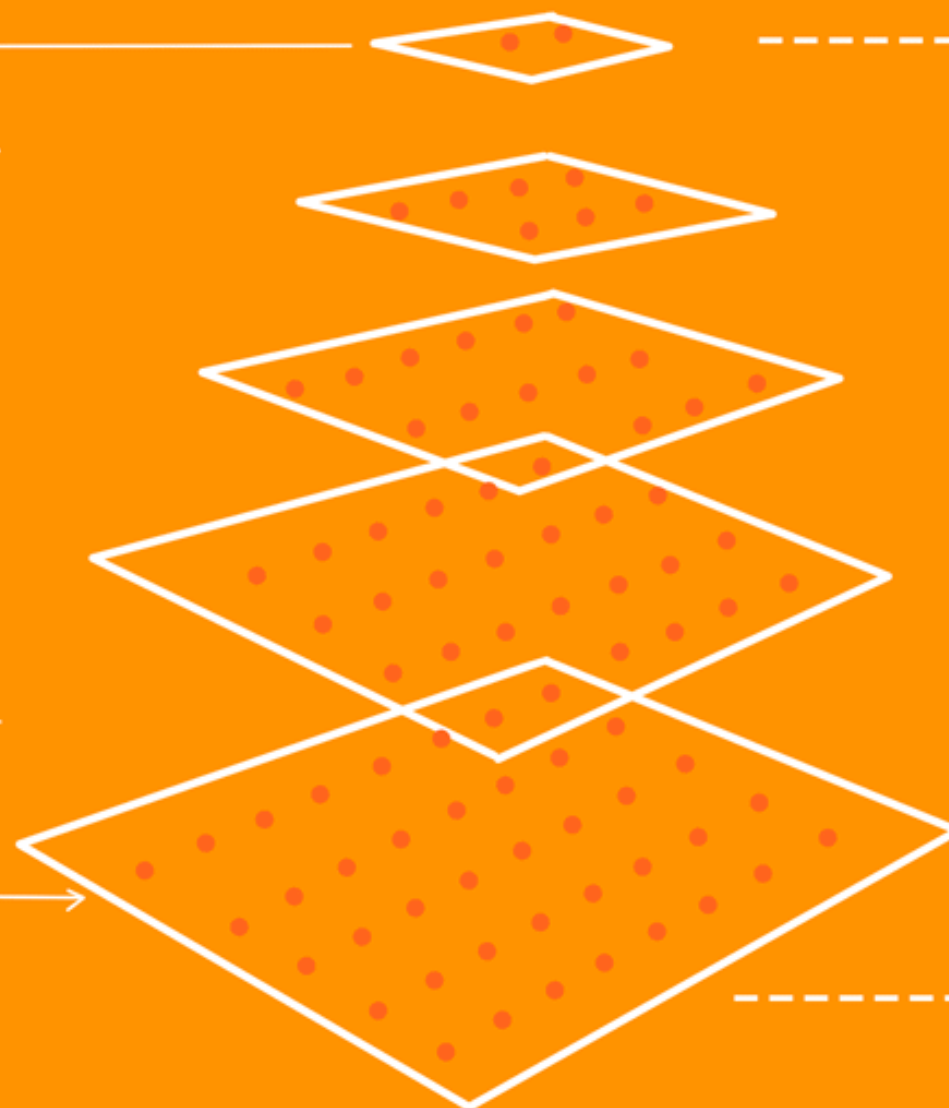
Output Layer

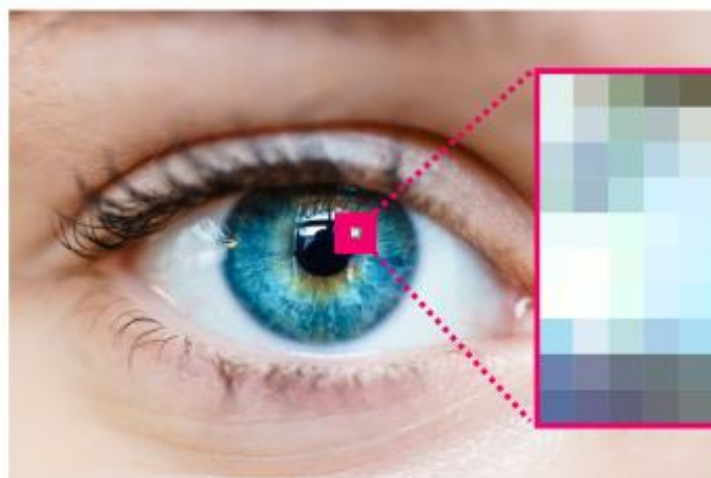
Activated
Neurons

Input Layer

DEEP
NEURAL
NETWORKS

MADRID MEETUP @Soygema





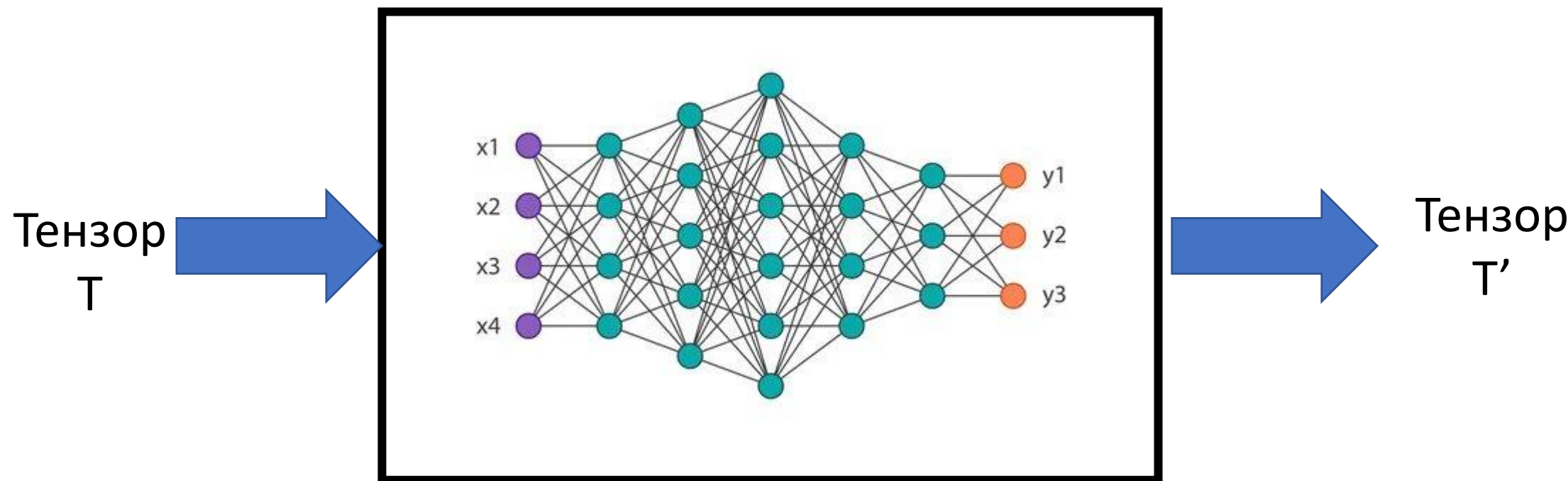
		233	188	137	96	90	95	63	73	73	82
	237	202	159	120	105	110	88	107	112	121	10
226	191	147	110	101	112	98	123	110	119	142	13
221	191	176	182	203	214	169	144	133	145	155	12
185	160	161	184	205	223	186	137	147	161	140	11
181	174	189	207	206	215	194	136	142	151	133	87
246	237	237	231	208	206	192	122	143	144	111	74
254	254	241	224	199	192	181	99	122	117	107	74
239	248	232	207	187	182	184	110	114	110	113	74
193	215	193	167	158	164	181	114	112	111	105	82
113	119	110	111	113	123	135	120	108	106	113	
93	97	91	103	107	111	122	112	104	114		

Что такое нейросеть?

переосмысливаем

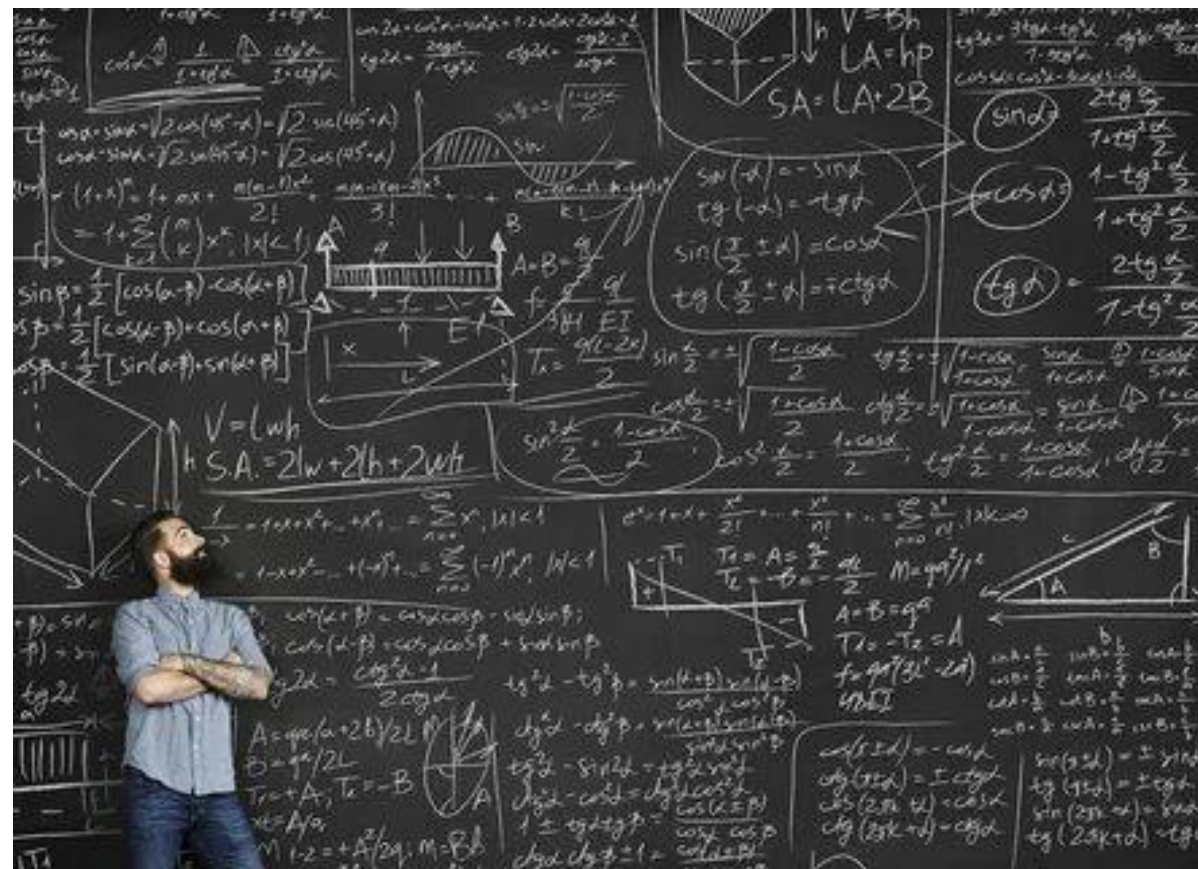
54

$$T' = f(T)$$



Нейросеть это функция, преобразующая один тензор в другой

Как работают инженеры и аналитики со всем ЭТИМ ХОЗЯЙСТВОМ?





**Пользуемся готовыми
библиотеками, используем
готовые функции и методы**

NumPy



Pandas





LEGO !





NumPy — это библиотека языка Python, добавляющая поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых (и очень быстрых) математических функций для операций с этими массивами.

Преимущества NumPy:

1. NumPy чрезвычайно быстр по сравнению с ядром Python благодаря интенсивному использованию расширений C.
2. Многие продвинутые библиотеки Python, такие как Scikit-Learn, Scipy и Keras, широко используют библиотеку NumPy. Поэтому, если вы планируете продолжить карьеру в области науки о данных или машинного обучения, NumPy - очень хороший инструмент для освоения.
3. NumPy поставляется с множеством встроенных функций, которые в ядре Python потребуют значительного количества настраиваемого кода.

Основным объектом NumPy является однородный многомерный массив элементов (как правило чисел) одного типа:

`numpy.ndarray`

`ndarray.ndim` - число измерений (чаще их называют "оси") массива.

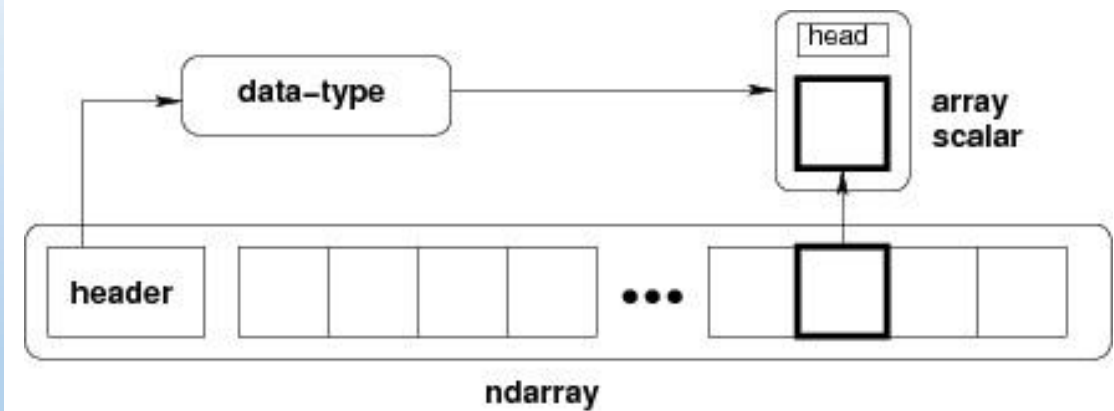
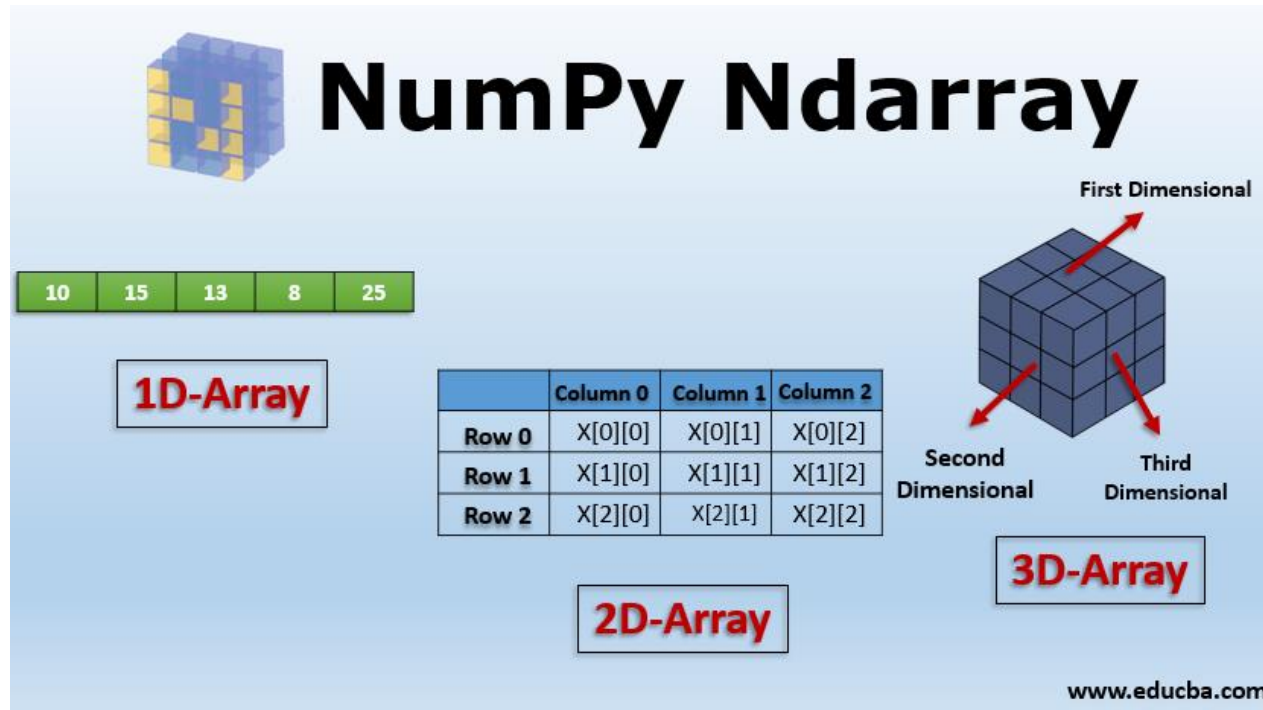
`ndarray.shape` - размеры массива, его форма. Это кортеж натуральных чисел, показывающий длину массива по каждой оси. Для матрицы из n строк и m столбцов, `shape` будет (n, m) . Число элементов кортежа `shape` равно `ndim`.

`ndarray.size` - количество элементов массива. Очевидно, равно произведению всех элементов атрибута `shape`.

`ndarray.dtype` - объект, описывающий тип элементов массива. Можно определить `dtype`, используя стандартные типы данных Python. NumPy здесь предоставляет целый букет возможностей, как встроенных, например: `bool_`, `character`, `int8`, `int16`, `int32`, `int64`, `float8`, `float16`, `float32`, `float64`, `complex64`, `object_`, так и возможность определить собственные типы данных, в том числе и составные.

`ndarray.itemsize` - размер каждого элемента массива в байтах.

`ndarray.data` - буфер, содержащий фактические элементы массива. Обычно не нужно использовать этот атрибут, так как обращаться к элементам массива проще всего с помощью индексов



Создание массива NumPy

```
import numpy as np
```

Command

NumPy Array

```
np.array([1,2,3])
```



1
2
3

```
np.arange(3)
```



0
1
2

```
np.ones(3)
```



1
1
1

```
np.zeros(3)
```



0
0
0

```
np.random.random(3)
```



0.5967
0.0606
0.2223

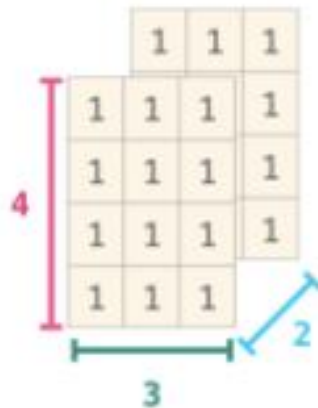
Создание тензоров NumPy

```
np.array([ [[1,2],[3,4]],  
          [[5,6],[7,8]] ])
```



		5	6
1	2		8
3	4		

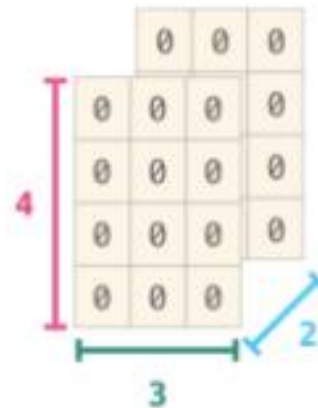
```
np.ones((4,3,2))
```



A 3D tensor visualization for `np.ones((4,3,2))`. It shows a stack of 4 planes, each with 3 rows and 2 columns. All cells contain the value 1. Dimensions are labeled: 4 (height), 3 (width), and 2 (depth).

	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

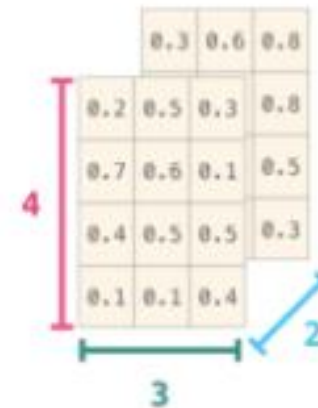
```
np.zeros((4,3,2))
```



A 3D tensor visualization for `np.zeros((4,3,2))`. It shows a stack of 4 planes, each with 3 rows and 2 columns. All cells contain the value 0. Dimensions are labeled: 4 (height), 3 (width), and 2 (depth).

	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

```
np.random.random((4,3,2))
```



A 3D tensor visualization for `np.random.random((4,3,2))`. It shows a stack of 4 planes, each with 3 rows and 2 columns. Cells contain random values between 0 and 1. Dimensions are labeled: 4 (height), 3 (width), and 2 (depth).

	0.3	0.6	0.8
0.2	0.5	0.3	0.8
0.7	0.6	0.1	0.5
0.4	0.5	0.5	0.3
0.1	0.1	0.4	

Арифметические операции над массивами NumPy

`data = np.array([1,2])`

data
1
2

`ones = np.ones(2)`

ones
1
1

`data + ones`

data
1
2

+

ones
1
1

=

2
3

`data - ones`

data
1
2

-

ones
1
1

=

0
1

`data * data`

data
1
2

*

data
1
2

=

1
4

`data / data`

data
1
2

/

data
1
2

=

1
1

`data * 1.6`

1
2

*

1.6

=

1
2

*

1.6
1.6

=

1.6
3.2

Арифметические операции над массивами NumPy

`np.arange(3)+5`

0	1	2
---	---	---

5	5	5
---	---	---

5	6	7
---	---	---

`np.ones((3,3))+np.arange(3)`

1	1	1
1	1	1
1	1	1

0	1	2
0	1	2
0	1	2

1	2	3
1	2	3
1	2	3

`np.ones((3,1))+np.arange(3)`

0	0	0
1	1	1
2	2	2

0	1	2
0	1	2
0	1	2

0	1	2
1	2	3
2	3	4

Индексация массивов NumPy

	data	data[0]	data[1]	data[0:2]	data[1:]
0	1	1		1	
1	2		2	2	2
2	3				3

Агрегирование массивов NumPy

data

1
2
3

$$\text{.max}() = 3$$

data

1
2
3

$$\text{.min}() = 1$$

data

1
2
3

$$\text{.sum}() = 6$$

data

1	2
3	4
5	6

$$\text{.max}() = 6$$

data

1	2
3	4
5	6

$$\text{.min}() = 1$$

data

1	2
3	4
5	6

$$\text{.sum}() = 21$$

`mean()` позволяет получить среднее арифметическое;
`prod()` выдает результат умножения всех элементов;
`std()` среднеквадратическое отклонение

Используя параметр `axis`, можно агрегировать не только все значения внутри матрицы, но и значения за столбцами или рядами.

data

1	2
5	3
4	6

$$\text{.max}(\text{axis}=0) = \begin{matrix} 1 & 2 \\ 5 & 3 \\ 4 & 6 \end{matrix} = \begin{matrix} 5 & 6 \end{matrix}$$

data

1	2
5	3
4	6

$$\text{.max}(\text{axis}=1) = \begin{matrix} 1 & 2 \\ 5 & 3 \\ 4 & 6 \end{matrix} = \begin{matrix} 2 \\ 5 \\ 6 \end{matrix}$$

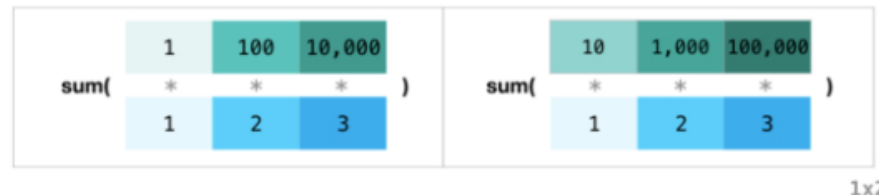
Скалярное произведение массивов NumPy

dot() Скалярное произведение NumPy

Главное различие с обычными **арифметическими операциями** здесь в том, что при умножении матриц используется скалярное произведение. В NumPy каждая матрица может использовать метод `dot()`. Он применяется для проведения скалярных операций с рассматриваемыми матрицами:



The diagram illustrates the dot product operation. On the left, a 1x3 matrix labeled 'data' contains the values [1, 2, 3]. A vertical bracket on its left indicates its height is 1, and a horizontal bracket below it indicates its width is 3. This matrix is followed by a dot product operation `.dot()` with a 3x2 matrix labeled 'powers_of_ten'. The 'powers_of_ten' matrix contains the values: [1, 10] in the first row, [100, 1,000] in the second row, and [10,000, 100,000] in the third row. The result of the dot product is a 1x2 matrix containing the values [30201, 302010]. Below the matrices, their dimensions are listed: 'Matrix dimensions: 1x3', '3x2', and '1x2'.



This diagram shows the row-wise calculation of the dot product. It consists of two 'sum()' operations. The first operation takes the first row of the 'powers_of_ten' matrix [1, 100, 10,000] and multiplies it element-wise with the 'data' matrix [1, 2, 3], then sums the results. The second operation takes the second row of the 'powers_of_ten' matrix [10, 1,000, 100,000] and multiplies it element-wise with the 'data' matrix [1, 2, 3], then sums the results. The final result is a 1x2 matrix.



This diagram shows the explicit scalar calculation of the dot product. It consists of two expressions separated by an equals sign. The first expression is `1*1 + 2*100 + 3*10,000`, which corresponds to the first row of the result. The second expression is `1*10 + 2*1,000 + 3*100,000`, which corresponds to the second row of the result. The final result is a 1x2 matrix containing the values [30201, 302010].

Транспонирование NumPy

`data`

1	2
3	4
5	6

`data.T`

1	3	5
2	4	6

Арифметические операторы NumPy

Оператор	Эквивалентная универсальная функция	Описание
+	<code>np.add</code>	Сложение (например, $1 + 1 = 2$)
-	<code>np.subtract</code>	Вычитание (например, $3 - 2 = 1$)
-	<code>np.negative</code>	Унарная операция изменения знака (например, -2)
*	<code>np.multiply</code>	Умножение (например, $2 * 3 = 6$)
/	<code>np.divide</code>	Деление (например, $3 / 2 = 1.5$)
//	<code>np.floor_divide</code>	Деление с округлением в меньшую сторону (например, $3 // 2 = 1$)
**	<code>np.power</code>	Возведение в степень (например, $2 ** 3 = 8$)
%	<code>np.mod</code>	Модуль/остаток (например, $9 \% 4 = 1$)

Функции агрегирования NumPy

Имя функции	NaN-безопасная версия	Описание
np.sum	np.nansum	Вычисляет сумму элементов
np.prod	np.nanprod	Вычисляет произведение элементов
np.mean	np.nanmean	Вычисляет среднее значение элементов
np.std	np.nanstd	Вычисляет стандартное отклонение
np.var	np.nanvar	Вычисляет дисперсию
np.min	np.nanmin	Вычисляет минимальное значение
np.max	np.nanmax	Вычисляет максимальное значение
np.argmin	np.nanargmin	Возвращает индекс минимального значения
np.argmax	np.nanargmax	Возвращает индекс максимального значения
np.median	np.nanmedian	Вычисляет медиану элементов
np.percentile	np.nanpercentile	Вычисляет квантили элементов
np.any	N/A	Проверяет, существуют ли элементы со значением true
np.all	N/A	Проверяет, все ли элементы имеют значение true