Deep Learning School

# Лекция 5

## Модель нейрона

# План лекции

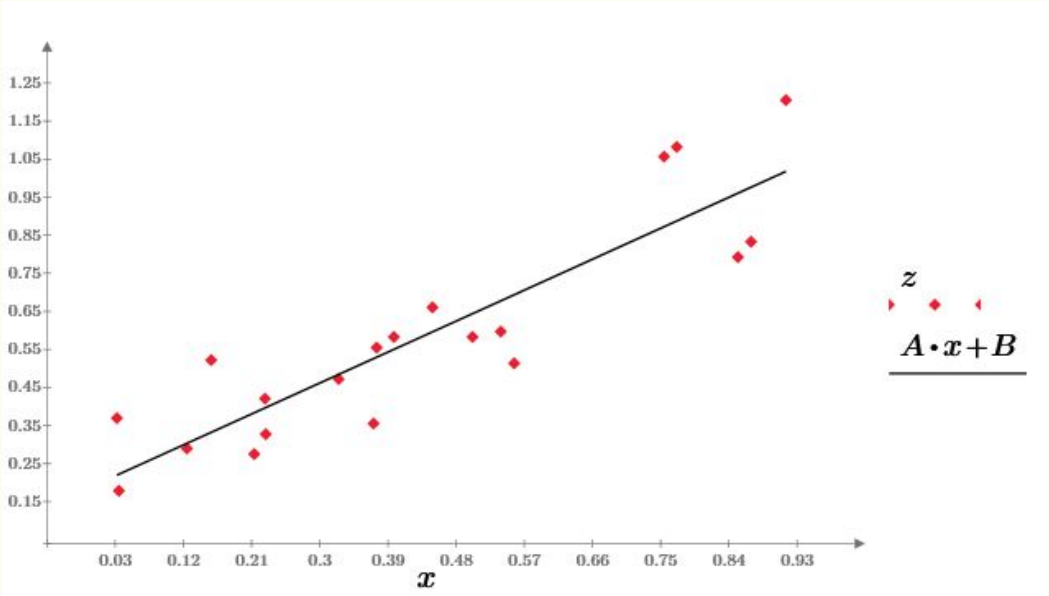
1. Recap линейных моделей
2. Recap градиентного спуска (GD)
3. SGD
4. Нейрон
5. Обучение нейрона
6. Функции активации

# Linear Regression Recap

# Одномерная линейная регрессия

(x,y) -- пары точек

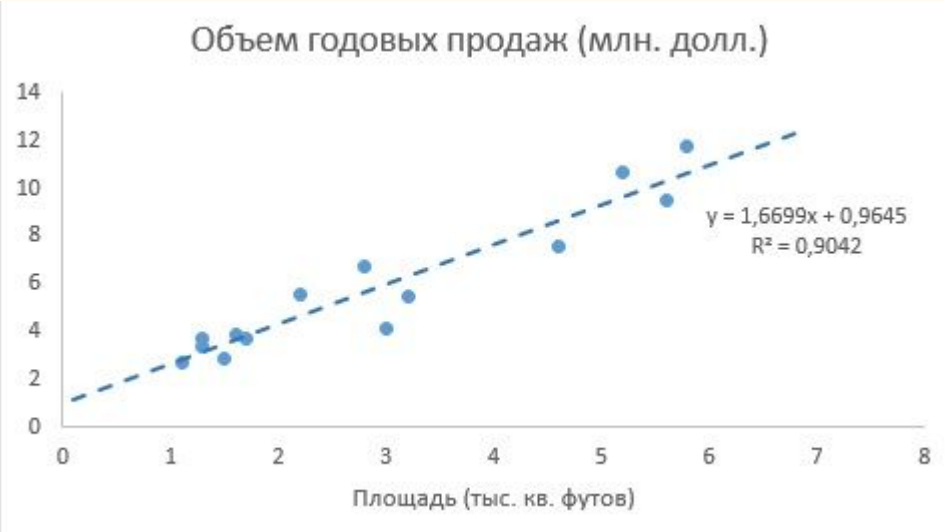
Задача: построить предсказания по x для неизвестных y в предположении, что y(x) -- линейная функция




$$y = Ax + B$$

# Пример

Наша цель — предсказать объем годовых продаж для всех новых магазинов, зная их размеры.




$$y = Ax + B$$

| | id | date | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | ... | grade | sqft_above | sqft_baseme |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 7129300520 | 20141013T000000 | 221900.00 | | 1.00 | 1180 | 5650 | 1.00 | 0 | 0 | ... | 7 | 1180 | 0 |
| 1 | 6414100192 | 20141209T000000 | 538000.00 | | 2.25 | 2570 | 7242 | 2.00 | 0 | 0 | ... | 7 | 2170 | 400 |
| 2 | 5631500400 | 20150225T000000 | 180000.00 | | 1.00 | 770 | 10000 | 1.00 | 0 | 0 | ... | 6 | 770 | 0 |
| 3 | 2487200875 | 20141209T000000 | 604000.00 | | 3.00 | 1960 | 5000 | 1.00 | 0 | 0 | ... | 7 | 1050 | 910 |
| 4 | 1954400510 | 20150218T000000 | 510000.00 | | 2.00 | 1680 | 8080 | 1.00 | 0 | 0 | ... | 8 | 1680 | 0 |

$$y = (y_1, \ldots, y_\ell.)$$

$$x = (x_1, \ldots, x_\ell)$$

$$w = (w_1, \ldots, w_d)$$

$$y^* = a(x) = w_0 + \sum_{j=1}^{d} w_j x_j$$

$$X = \{(x_1, y_1), \ldots, (x_\ell, y_\ell)\}$$

# Матричная запись

$$\begin{aligned} y_1 &= ax_1 + b + \varepsilon_1 \\ y_2 &= ax_2 + b + \varepsilon_2 \\ &\cdots\cdots\cdots\cdots \\ y_n &= ax_n + b + \varepsilon_n \end{aligned}$$

$$\vec{y} = X\vec{w} + \epsilon,$$

# Фиктивная переменная (случай 2х параметров, n наблюдений)

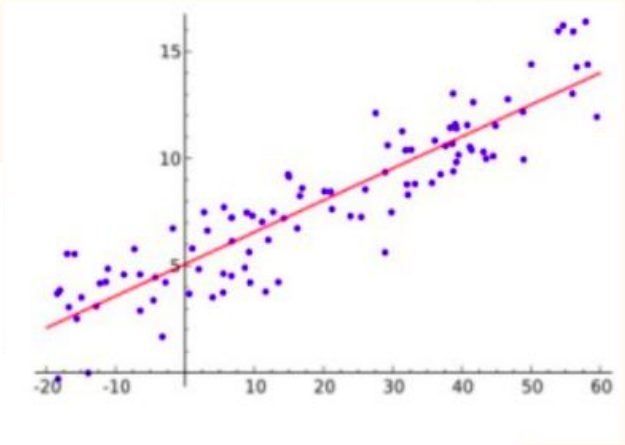
$$X\omega = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ .. & .. \\ 1 & x_n \end{pmatrix} \begin{pmatrix} b \\ a \end{pmatrix} = \begin{pmatrix} ax_1 + b \\ ax_2 + b \\ \cdots\cdots\cdots \\ ax_n + b \end{pmatrix}$$

$$\vec{y} = X\vec{w} + \epsilon,$$

$$\omega = \begin{pmatrix} b \\ a \end{pmatrix}$$

$$\mathbf{x} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ .. & .. \\ 1 & x_n \end{pmatrix}$$

# Линейная регрессия




$$\vec{y} = X\vec{w} + \epsilon,$$

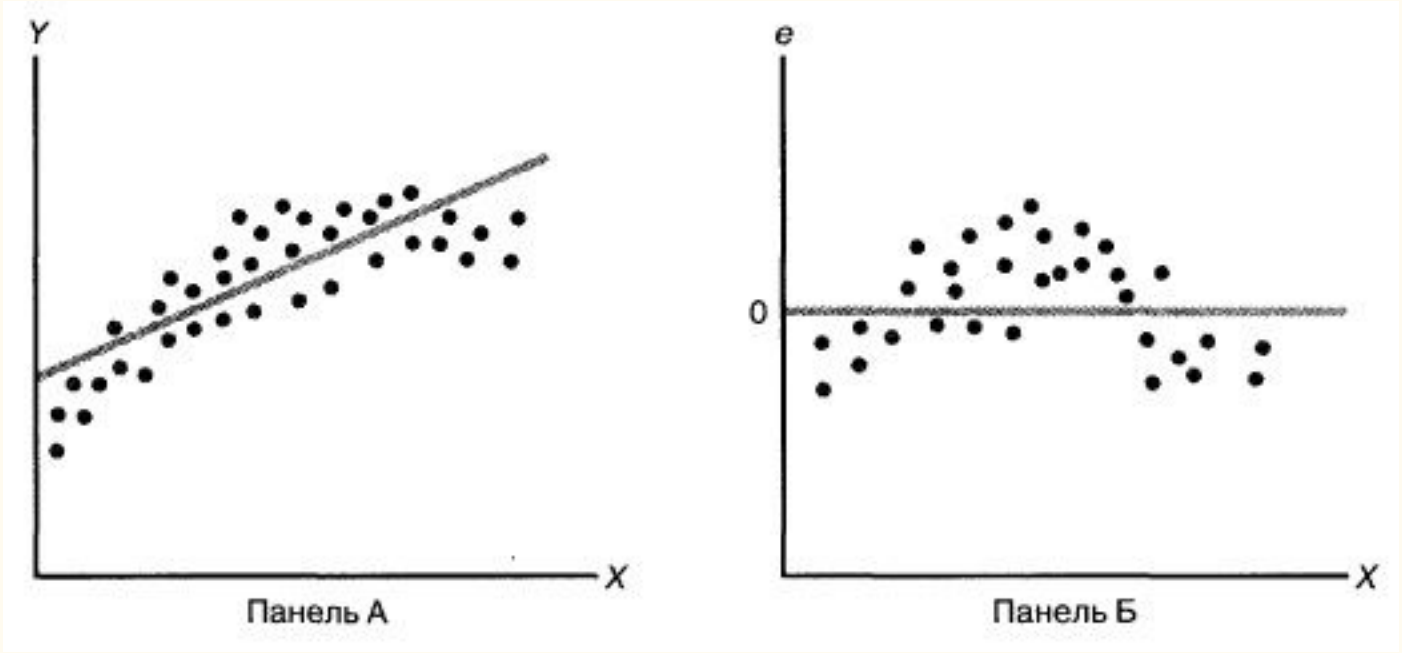
где

- $\vec{y} \in \mathbb{R}^n$ – объясняемая (или целевая) переменная;
- $w$ – вектор параметров модели (в машинном обучении эти параметры часто называют весами);
- $X$ – матрица наблюдений и признаков размерности $n$ строк на $m + 1$ столбцов (включая фиктивную единичную колонку слева) с полным рангом по столбцам: $\mathrm{rank}\,(X) = m + 1$;
- $\epsilon$ – случайная переменная, соответствующая случайной, непрогнозируемой ошибке модели.

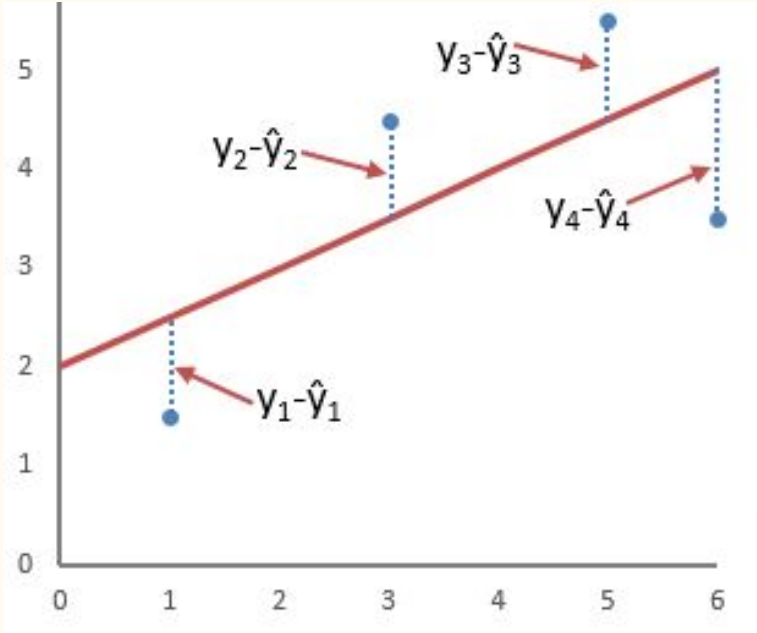
https://habr.com/company/ods/blog/323890/

# Gradient descent Recap

# Построение прогноза



# Метод наименьших квадратов (одномерный случай)




Функция потерь — *квадратичная*:

$$\mathscr{L}(a, y) = (a - y)^2$$

Метод обучения — *метод наименьших квадратов*:

$$Q(w) = \sum_{i=1}^{\ell} \big(a(x_i, w) - y_i\big)^2 \to \min_w$$

# МНК (многомерный случай)

$$Q(\alpha, X^\ell) = \sum_{i=1}^{\ell} \big(f(x_i, \alpha) - y_i\big)^2 = \|F\alpha - y\|^2 \to \min_\alpha .$$

$$f(x, \alpha) = \sum_{j=1}^{n} \alpha_j f_j(x), \qquad \alpha \in \mathbb{R}^n.$$

Матричные обозначения:

$$\underset{\ell\times n}{F} = \begin{pmatrix} f_1(x_1) & \dots & f_n(x_1) \\ \dots & \dots & \dots \\ f_1(x_\ell) & \dots & f_n(x_\ell) \end{pmatrix}, \quad \underset{\ell\times 1}{y} = \begin{pmatrix} y_1 \\ \dots \\ y_\ell \end{pmatrix}, \quad \underset{n\times 1}{\alpha} = \begin{pmatrix} \alpha_1 \\ \dots \\ \alpha_n \end{pmatrix}.$$

# Общий случай

$$Q(\alpha, X^l) = \sum_{i=1}^{l} L(f(x_i, w), y_i) \to min$$

# Метод градиентного спуска

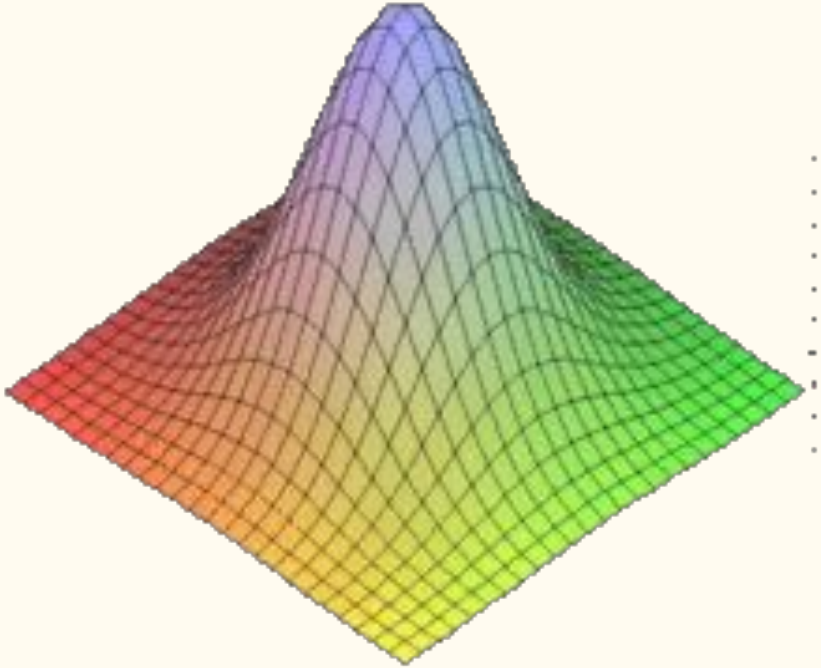
$$Q(\alpha, X^l) = \sum_{i=1}^{l} L(f(x_i, w), y_i) \to min$$

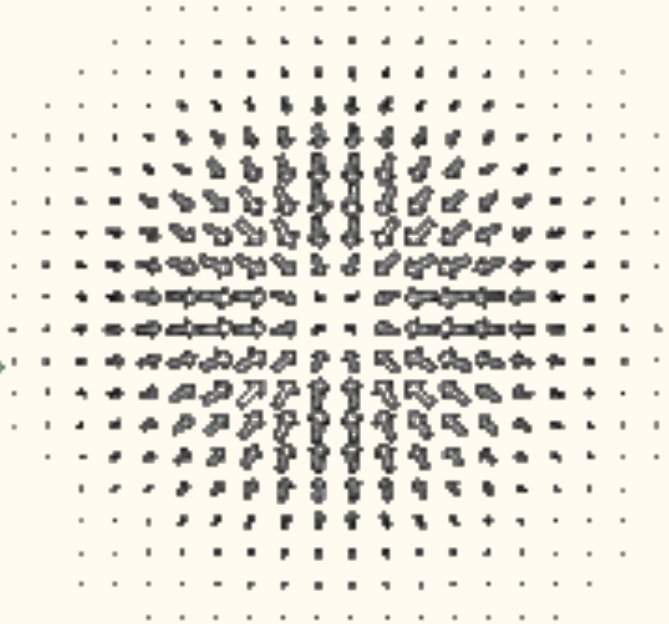
Нужно подобрать вектор параметров

Минимизируем ошибку путем минимизации функции потерь

# Градиент

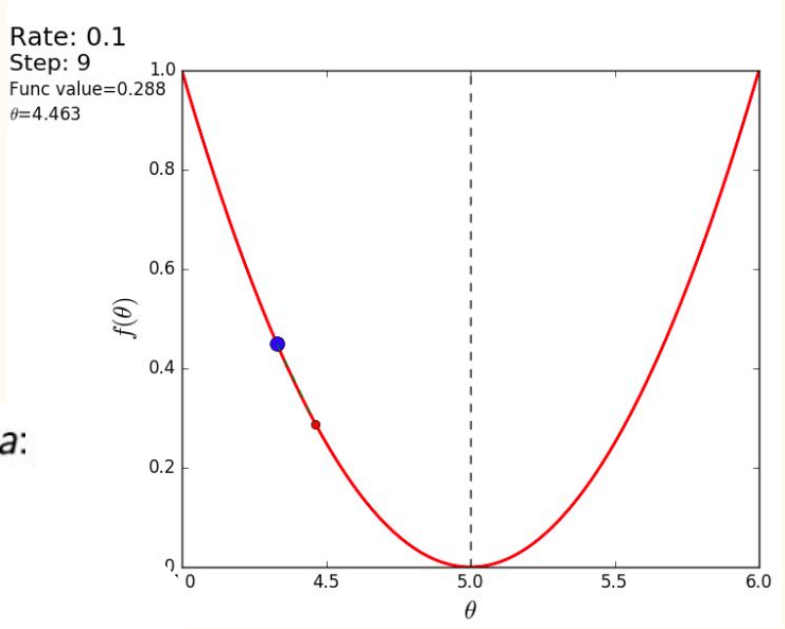
$\nabla\varphi$ $\operatorname{grad}\varphi$





# Градиентный спуск (GD)

$$Q(\alpha, X^l) = \sum_{i=1}^{l} L(f(x_i, w), y_i) \to min$$




Численная минимизация методом *градиентного спуска*:

$w^{(0)} :=$ начальное приближение:

$$w^{(t+1)} := w^{(t)} - h \cdot \nabla Q(w^{(t)}),$$

где $h$ — *градиентный шаг*, называемый также *темпом обучения*

$$w^{(t+1)} := w^{(t)} - h \sum_{i=1}^{\ell} \nabla \mathscr{L}_i\left(w^{(t)}\right).$$

# “Спуск ночью с фонариком в горах”

# Градиентный спуск **(GD)**

1. Складываем градиенты для каждого примера из выборки
2. Обновляем веса после вычисления производных для **всех** элементов выборки

*Алгоритм:*

Повторять, пока не сойдется:

$$w_i = w_i + \alpha \sum_{i=1}^{n} \frac{dL(f(x_i, w), y_i)}{dw_i}$$

# Стохастический градиентный спуск

# Стохастический градиентный спуск **(SGD)**

1. Вычисляем градиент для одного примера из выборки
2. Обновляем веса каждый раз после вычисления производных для **одного** элемента выборки

## *Алгоритм **GD:***

Повторять, пока не сойдется:

$$w_i = w_i + \alpha \sum_{i=1}^{n} \frac{dL(f(x_i, w), y_i)}{dw_i}$$

## *Алгоритм **SGD:***

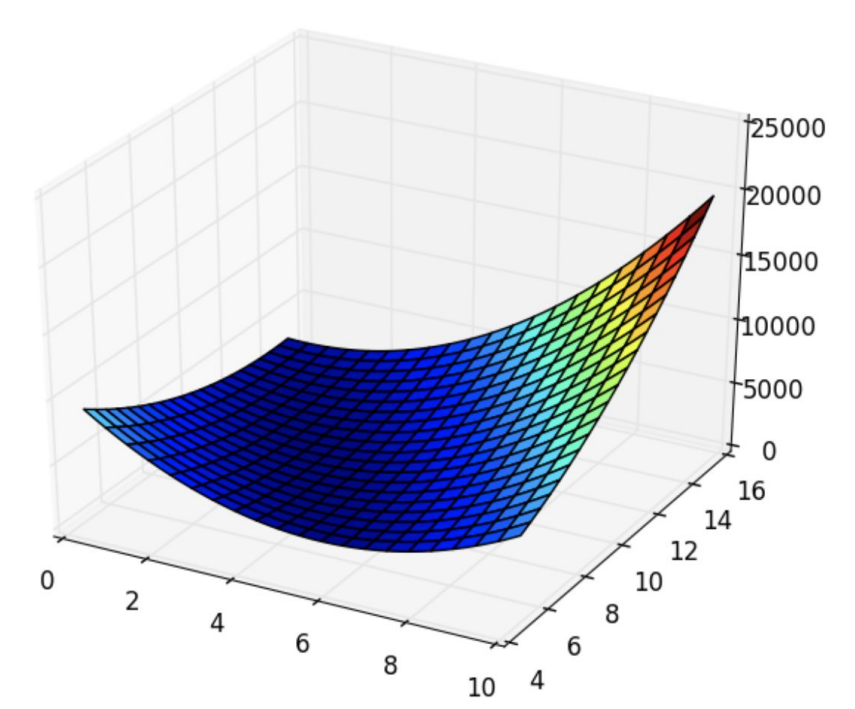
Повторять, пока не сойдется:

for i in range(len(X_train)):

$$w_i = w_i + \alpha \frac{dL(f(x_i, w), y_i)}{dw_i}$$

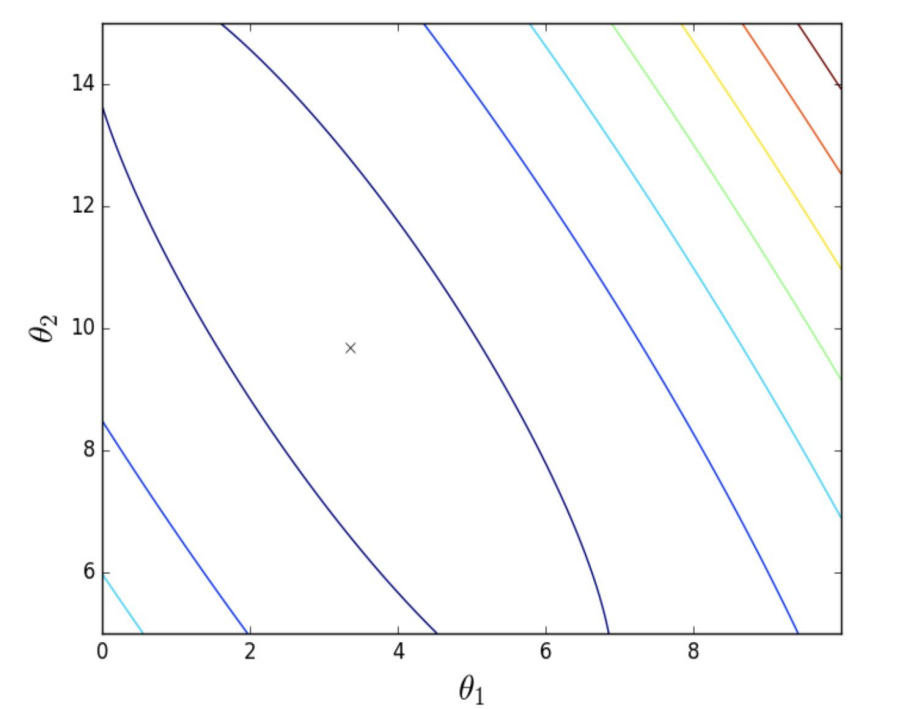
# GD vs SGD

Вид функции потерь:

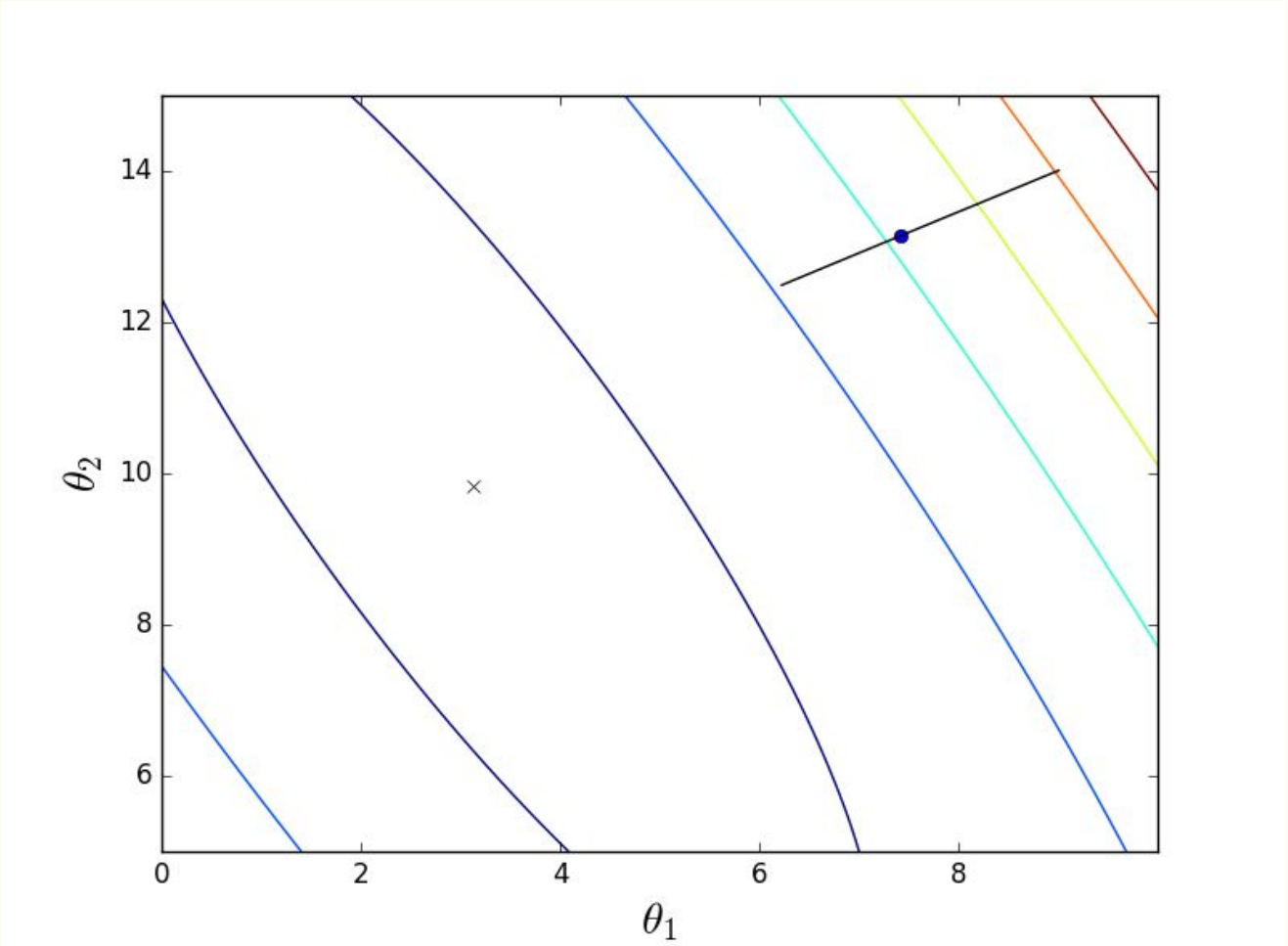



Вид сверху:

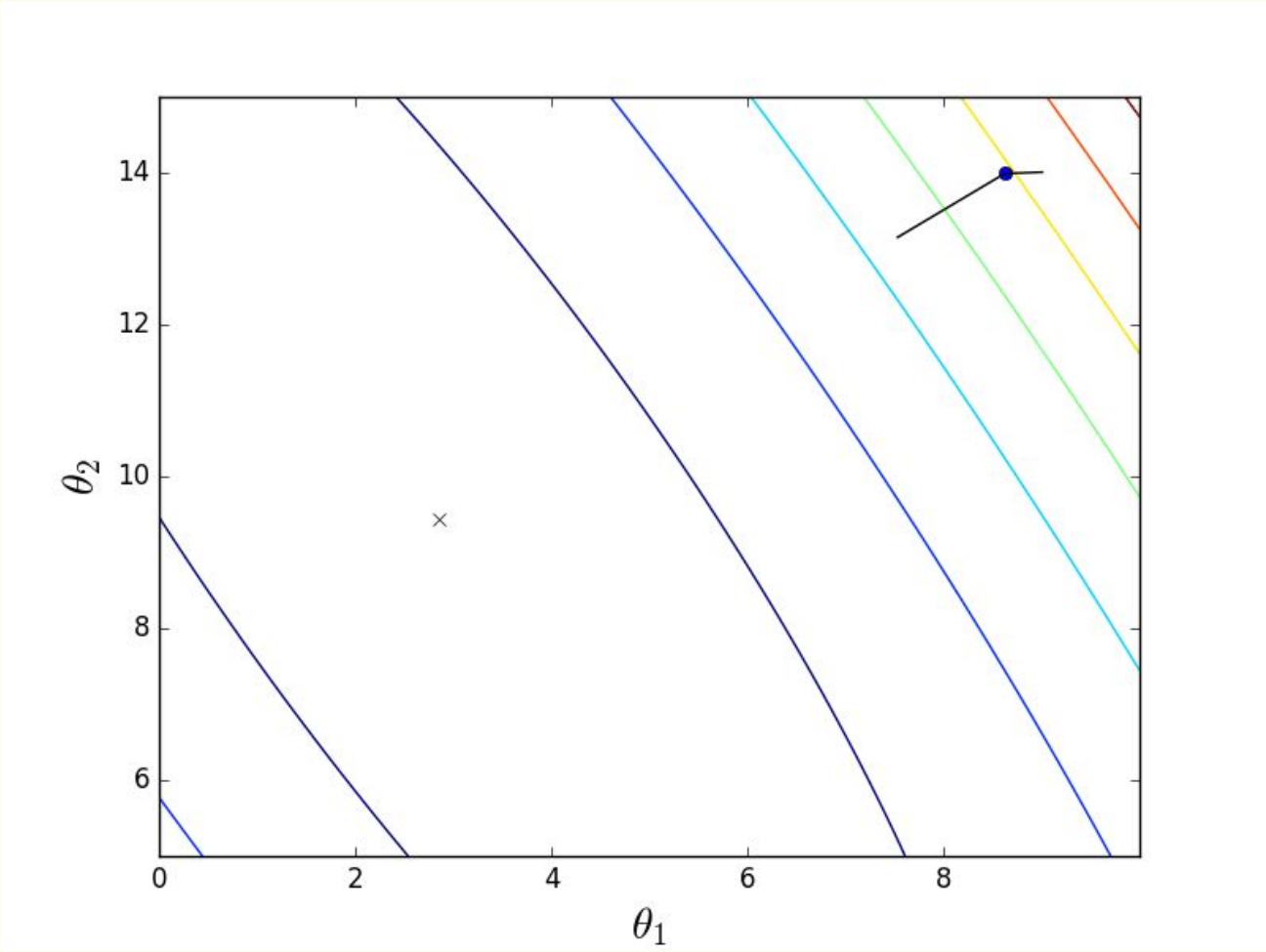
крестик -- точка минимума, куда должен стремиться алгоритм



# GD



# SGD



# GD vs SGD

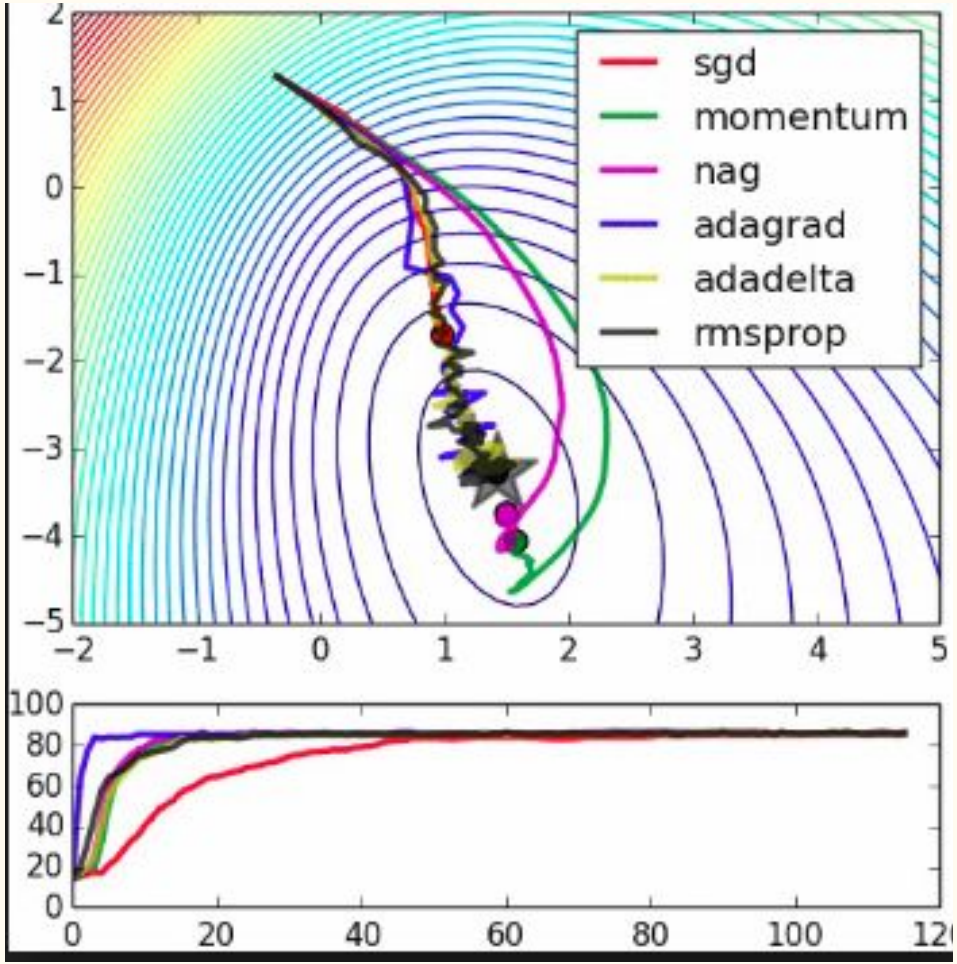
- SGD быстрее обновляет веса, но требует больше шагов для сходимости
- SGD хорошо работает на функциях с большим количеством локальных минимумов -- каждый шаг есть шанс, что очреденое значение “выбьет” из локальной ямы и конечное решение будет более оптимальным, чем для GD
- GD хорош для строго выпуклых функций -- он уверенно стремится к точке локального минимума

# GD vs SGD

Компромисс?

## **Mini-batch** gradient descent

Каждый раз обновлять веса по сумме градиентов по небольшой подвыборке.


sgd
momentum
nag
adagrad
adadelta
rmsprop
2
1
0
−1
−2
−3
−4
−5
−2
−1
0
1
2
3
4
5
100
80
60
40
20
0
0
20
40
60
80
100

# Нейрон

# Линейный классификатор

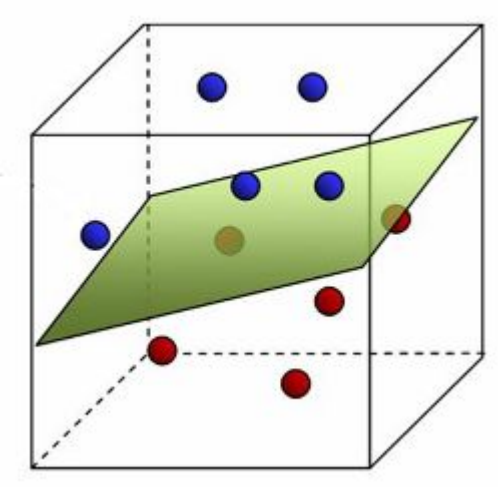
Как из линейной регрессии сделать линейную классификацию?

$$a(x) = w_0 + \sum_{j=1}^{d} w_j x_j$$

# Линейный классификатор

Как из линейной регрессии сделать линейную классификацию?

$$a(x) = w_0 + \sum_{j=1}^{d} w_j x_j$$



$$a(x, w) = \text{sign}\langle x, w\rangle = \text{sign} \sum_{j=1}^{n} w_j f_j(x)$$

# Линейный классификатор как модель нейрона

Линейная модель нейрона МакКаллока-Питтса [1943]:

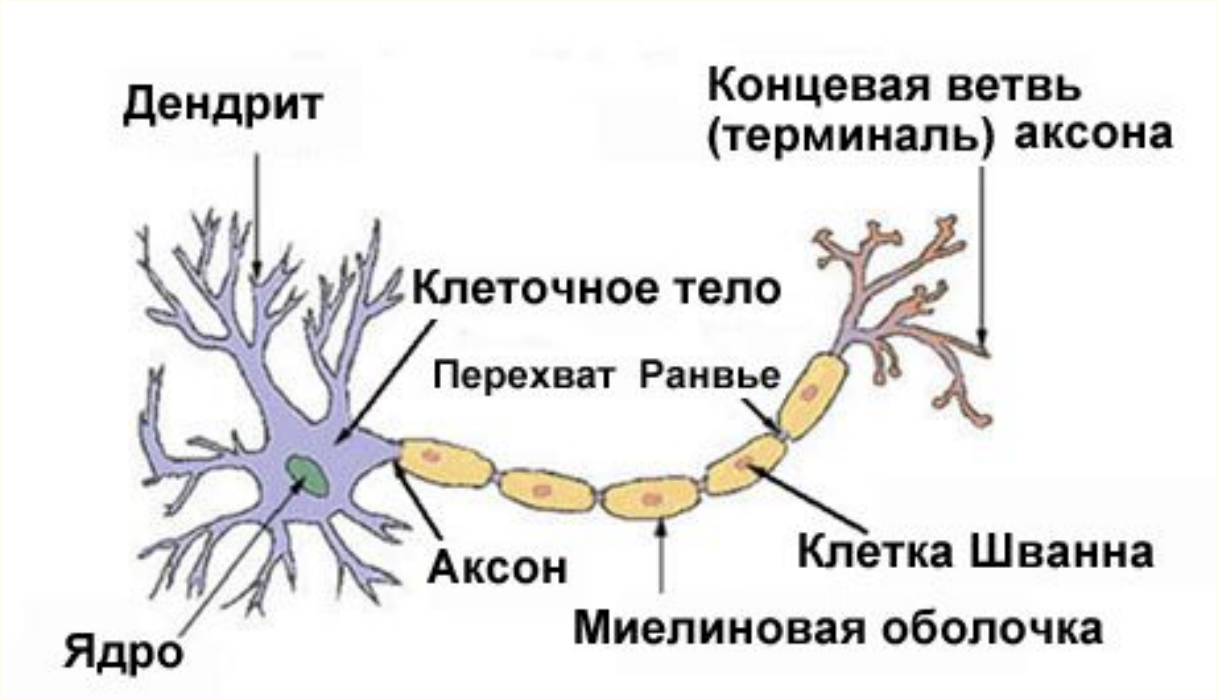
$$a(x, w) = \sigma(\langle w, x \rangle) = \sigma\left(\sum_{j=1}^{n} w_j f_j(x) - w_0\right).$$

$\sigma(z)$ — функция активации (например, sign),

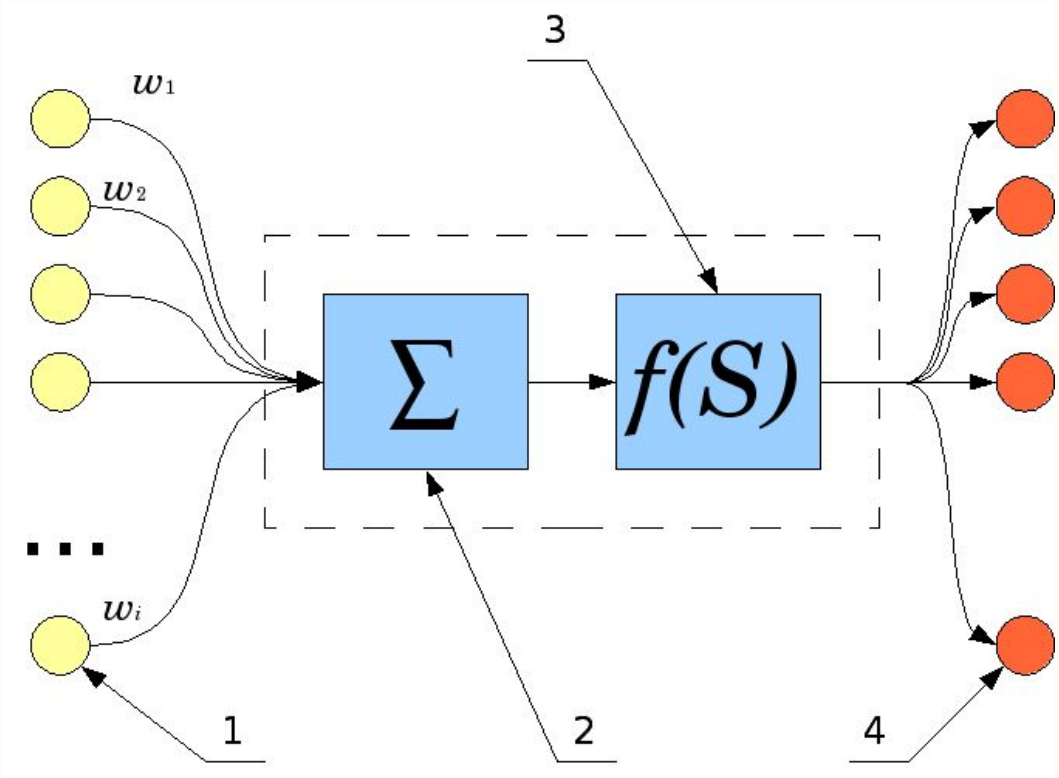
$w_j$ — весовые коэффициенты синаптических связей,

$w_0$ — порог активации,

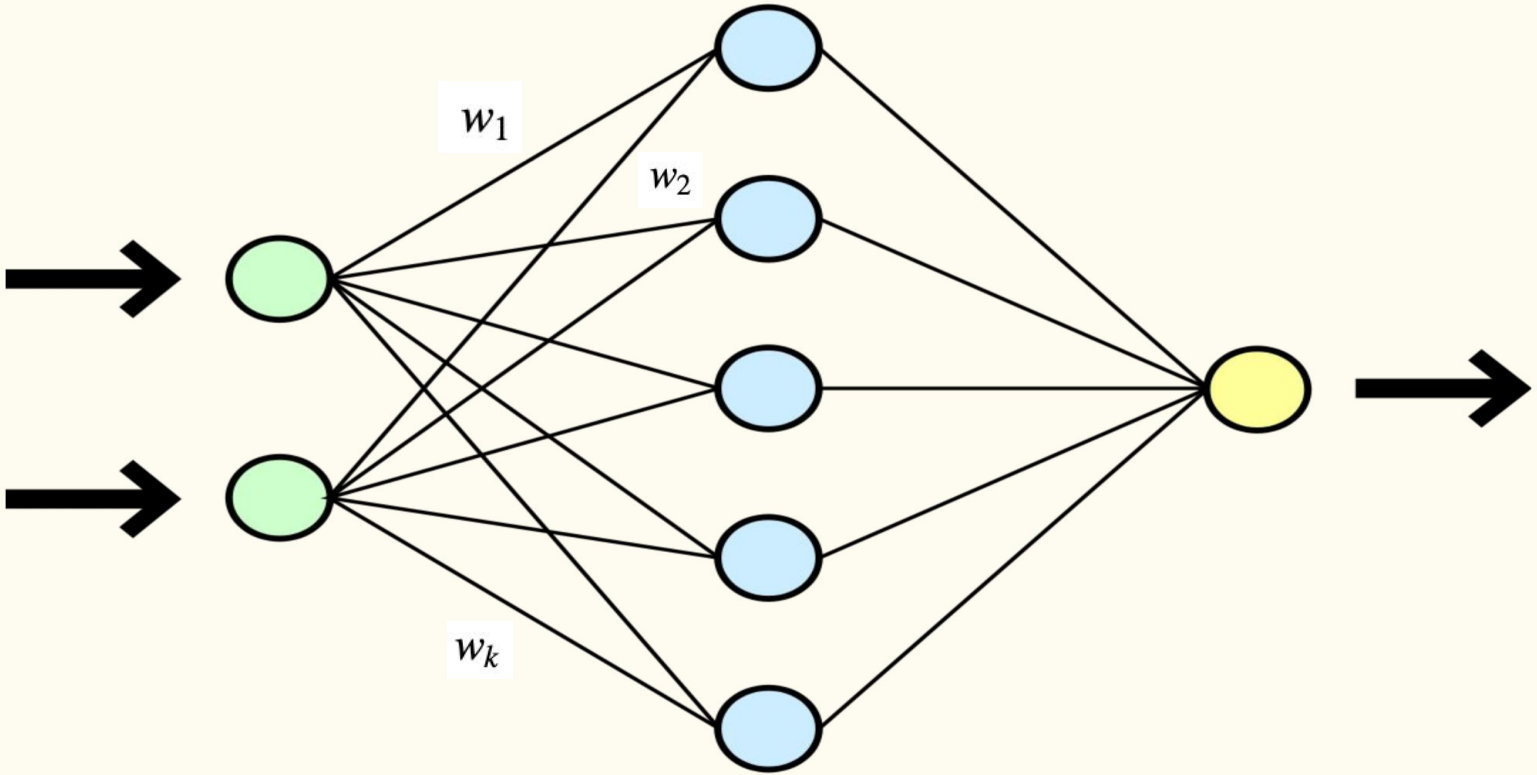
# Биологический нейрон человека



# Искусственный нейрон



# Нейронная сеть



# Нейрон: обучение

Обучение градиентным спуском:

$$a(x, w) = \sigma(\langle w, x \rangle) = \sigma\left(\sum_{j=1}^{n} w_j f_j(x) - w_0\right)$$

$$w_i = w_i + \alpha \frac{dL(f(x_i, w), y_i)}{dw_i}$$

# Нейрон: обучение

Пример с $L(f(x_i, w), y_i) = MSE$

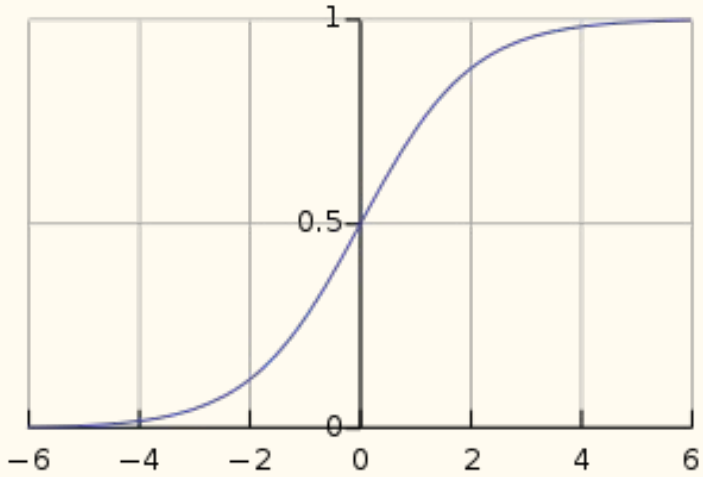
$$\frac{dL(f(x_i, w), y_i)}{dw_i} = 2(f(x_i, w) - y_i)\frac{df(x_i, w)}{dw_i} =$$

$$= 2(f(x_i, w) - y_i)\frac{d\sigma}{d(w_i f_i + b)}\frac{d(w_i f_i + b)}{dw_i}$$

# Нейрон: функции активации

- Sigmoid: $$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Область значений: [0, 1]



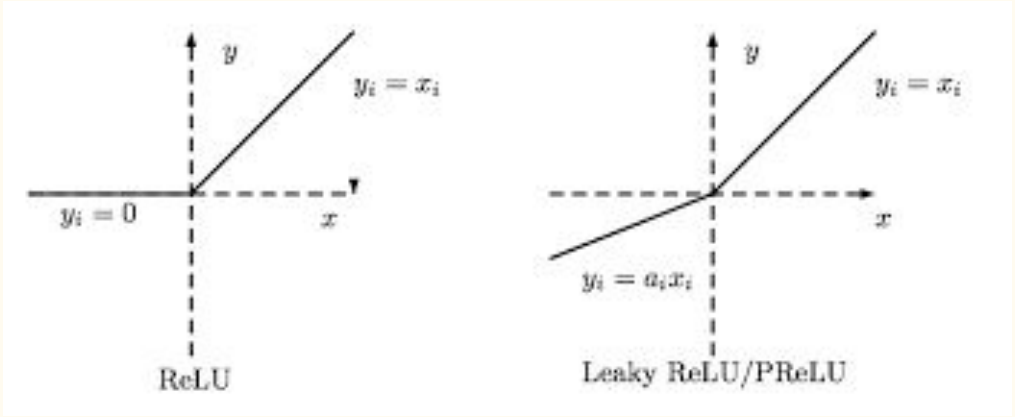

$$\frac{d\sigma}{dx} = \sigma(x) \cdot (1 - \sigma(x))$$

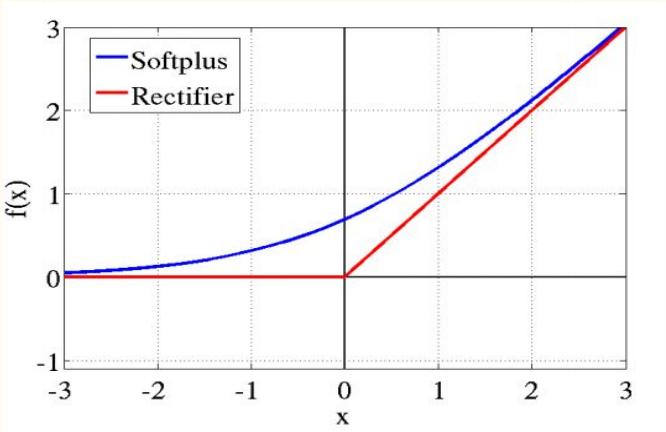
Для каких задач применима?

# Нейрон: функции активации

Другие функции активации:

- Relu
- LeakyRelu
- Softmax




- Tanh
- Elu
- ...

# The End