

# Практический анализ данных и машинное обучение: искусственные нейронные сети

Ульянкин Филипп и Соловей Влад

2 февраля 2019 г.

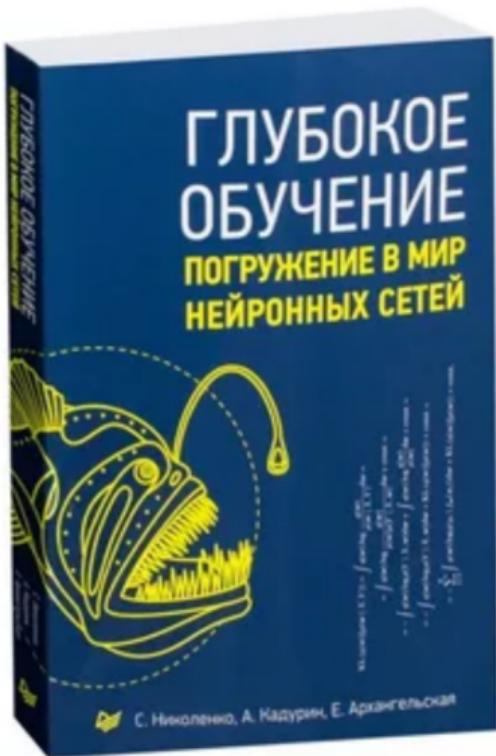
Введение в Нейросети

Я

# Структура курса

- Введение в нейросетки
- Как обучаются нейросетки, эвристики, используемые при обучении
- Свёрточные нейронные сети
- Сети прямого распространения в анализе текстов
- Рекурентные нейросетки
- Обзор современных нейросетевых конструкций (GANы, автопереводы, нейробайесовский подход и тп)

# Что почитать про нейронки



# Что посмотреть про нейронки

- Если ничего не знаете про машинное обучение, смотрите вводный курс от Яндекса и МФТИ:  
<https://www.coursera.org/specializations/machine-learning-data-analysis>
- Серия лекций техносферы:  
<https://www.youtube.com/watch?v=Am82yvUSwRE>
- Введение в нейронные сети от Andrew Ng:  
<https://www.coursera.org/instructor/andrewng>
- Advanced ML от Яндекса: <https://www.coursera.org/specializations/aml>

# Agenda

- Немного истории
- Напоминание о линейных моделях, переобучении, регуляризаторах и кросс-валидации
- Всё, что вы хотели знать о градиентном спуске, но боялись спросить
- От линейных моделей к нейросеткам
- Нейросетки — конструктор Lego

# История нейросеток

# Первый формальный нейрон, 1943 год



Уолтер Питтс



Уоррен Маккалок

# Первый искусственная нейросеть, 1958 год

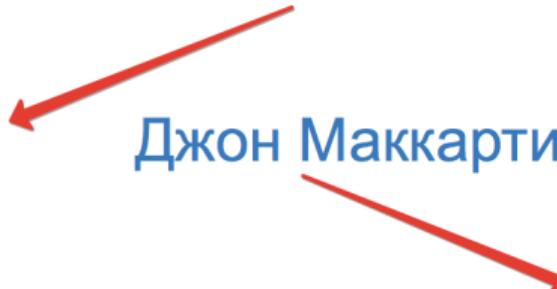


Фрэнк Розенблатт





Марвин Минский



Джон Маккарти

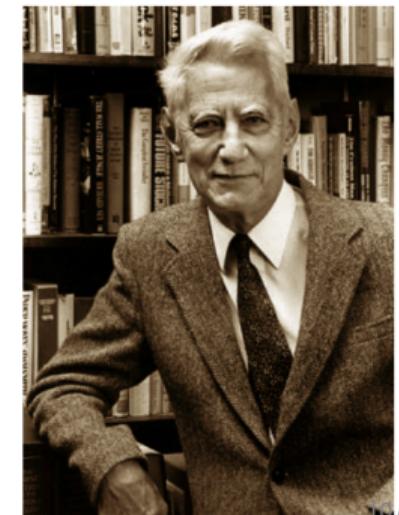


1956 год

Натаниэль Рочестер

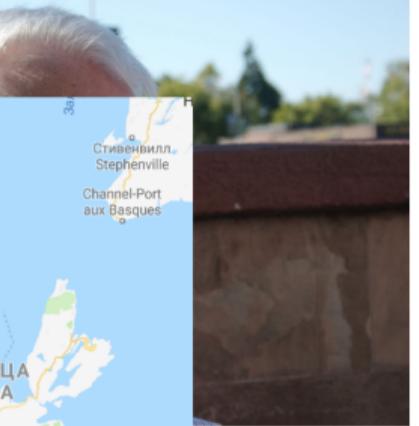
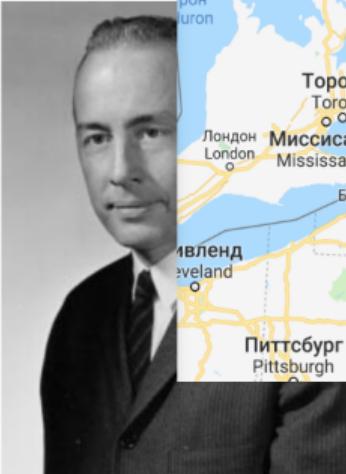
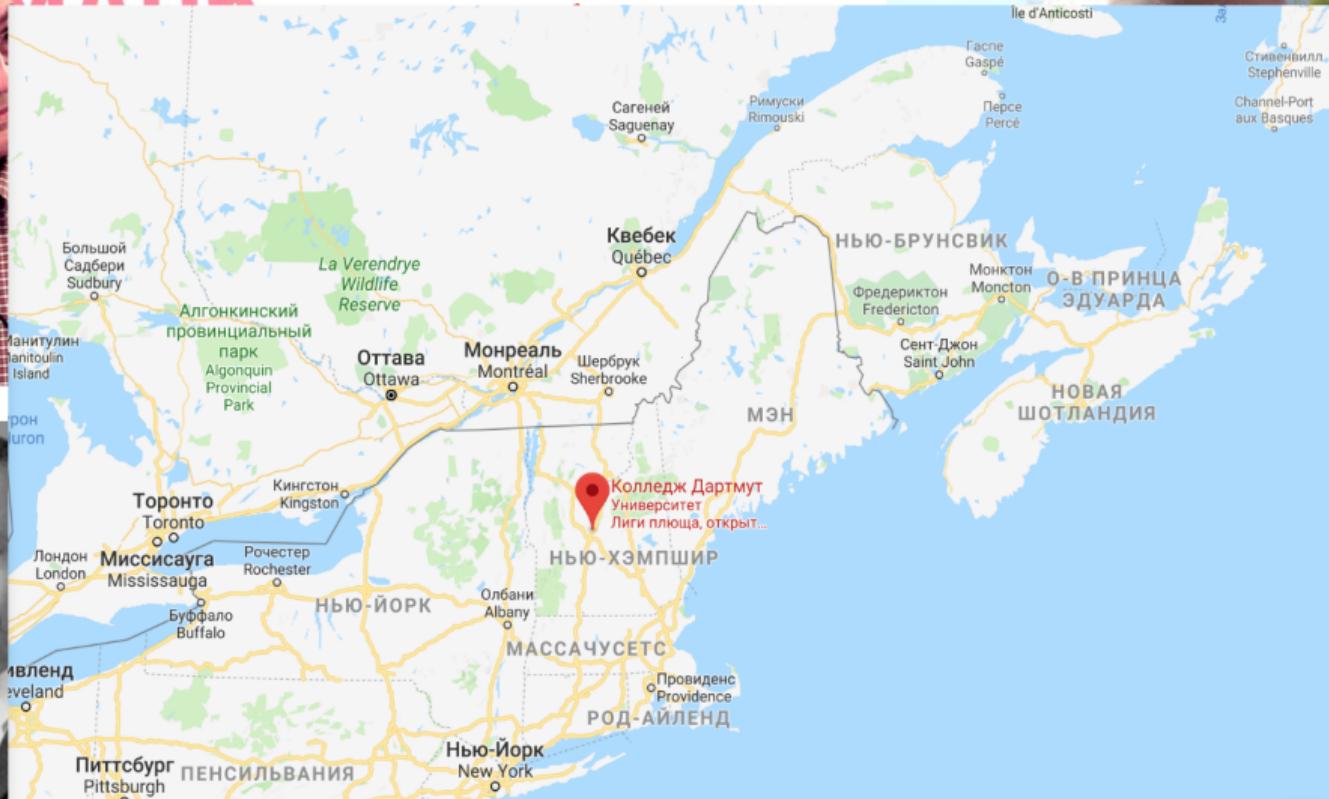


Клод Шеннон



ИНГИ  
МАТИК

# Марвин Минский



# Зима близко

- 1956 – Дартмунтский семинар, море оптимизма
- 1958 – Перцептрон Розенблатта
- середина 1960-х – провал крупного проекта по машинному переводу с русского на английский и наоборот
- 1969 – Марвин Минский и Сеймур Пейперт опубликовали книгу «Перцептроны» с критикой



memegenerator.net

# Оттепель

- 1970-е — Расцвет экспертных систем, принимающих решения на основе большого числа правил и знаний о предметной области
- MYCIN накопила около 600 правил для идентификации вирусных бактерий и выдачи подходящего метода лечения (угадывала в 69% случаев, лучше любого начинающего врача)
- 1980-е — появилось много разных архитектур
- 1980-е — backpropagation позволил обучать сети за линейное время
- Ренессанс нейронных сетей

# Зима близко

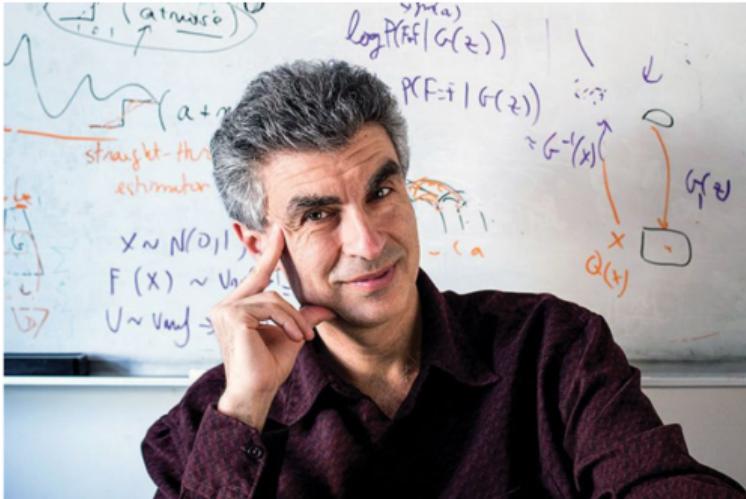
- Новая волна оптимизма
- 1986 — один из первых AI-отделов экономил компании DEC около 10 миллионов долларов в год
- Завышенные ожидания снова лопнули
- 1990-е — ударными темпами развивается классическое машинное обучение



# 2005-2006: начало революции



Джеффри Хинтон  
университет Торонто

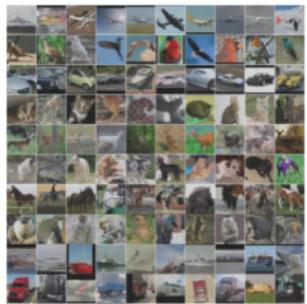


Йошуа Бенджи  
университет Монреяля

# Революция

- 2005-2006 — группы Хинтона и Бенджи научились обучать глубокие нейросетки
- Накопилось больше данных! Огромные данные!
- Компьютеры стали на порядки мощнее! Появились крутые GPU!
- На больших данных и мощностях заработали старые архитектуры
- Появились новые алгоритмы, эвристики и подходы
- Ящик Пандоры открыт!

# ImageNet

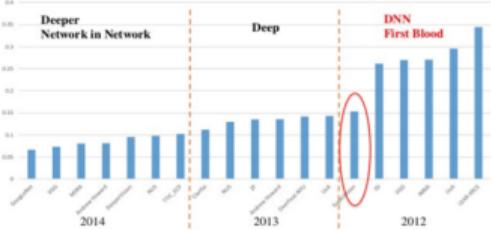


AlphaGo

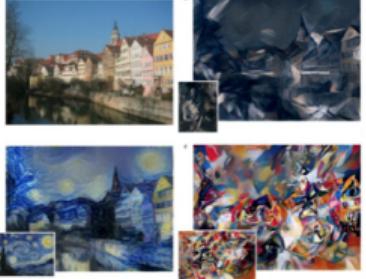
4:1



ImageNet Classification error throughout years and groups



Пабло Пикассо



Винсент Ван Гог



Василий Кандинский



# Линейные модели

# Линейная регрессия (постановка)

- $y \in R$  — целевая переменная (рейтинг фильма, стоимость квартиры, валютный курс и тп)
- $X$  — признаки
- Модель:

$$y = w_0 + w_1x_1 + w_2x_2 + \dots + w_kx_k$$

- Нужно оценить  $k + 1$  параметр

# Линейная регрессия (векторная форма)

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

Модель:

$$y = Xw$$

Прогноз:

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

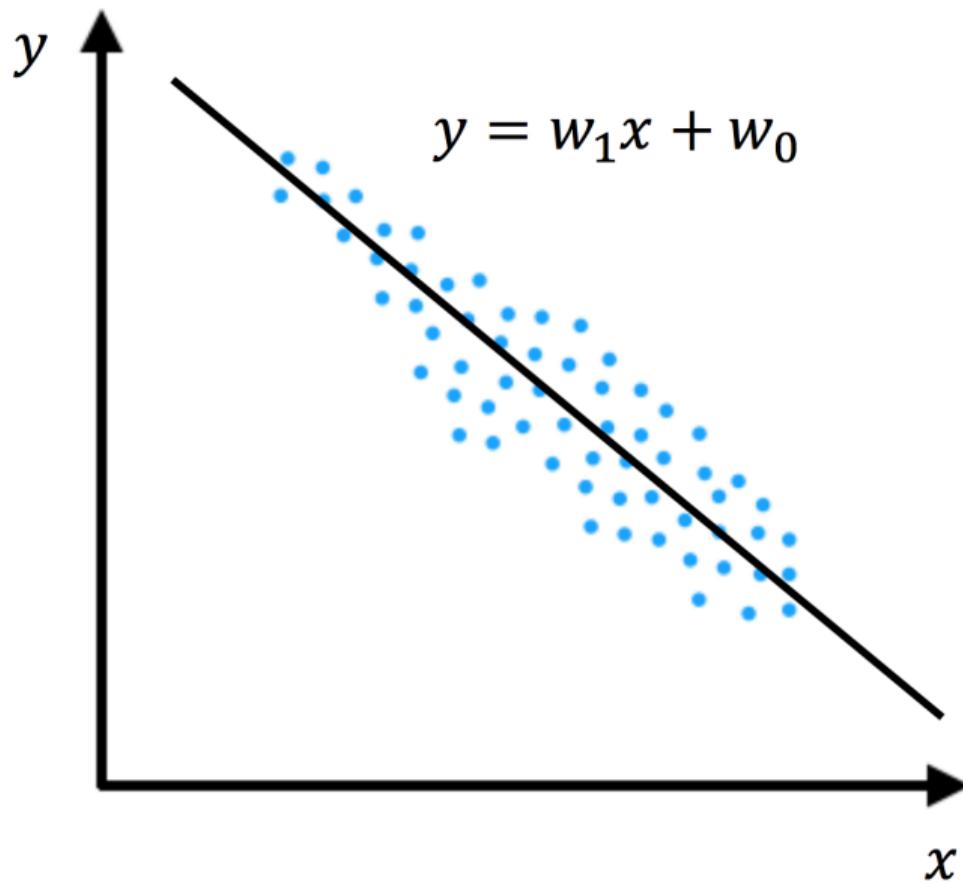
# Линейная регрессия (функция потерь)

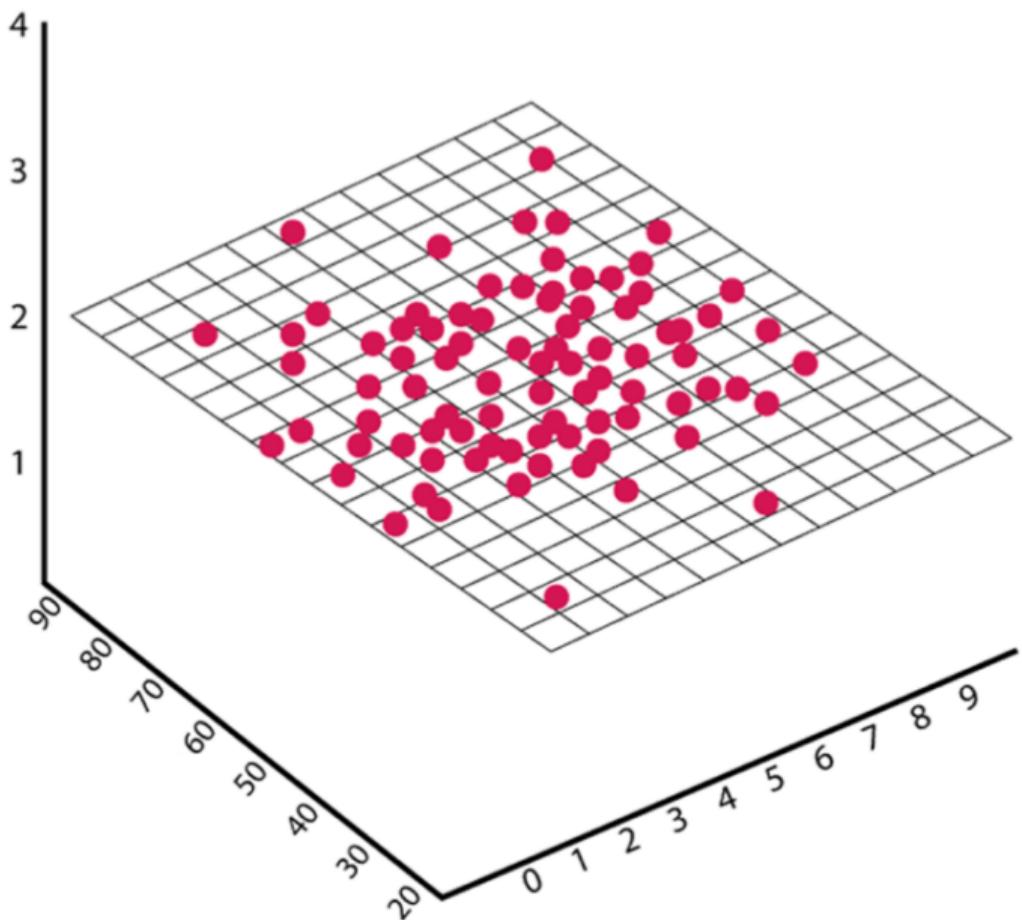
Как тренировать модель?

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

Существует аналитическое решение:

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





## Резюме

- Линейная регрессия очень простая для обучения модель
- $MSE$  можно использовать как функцию потерь для линейной регрессии, в такой ситуации существует аналитическое решение
- Не для всех функция потерь бывает аналитическое решение, пример:  $MAE$

$$MAE = \frac{1}{n} \sum_{i=1}^n |w^T x_i - y_i|$$

- Поэтому на практике для обучения не используют аналитическое решение, выбирая более масшабируемые методы

# Классификация

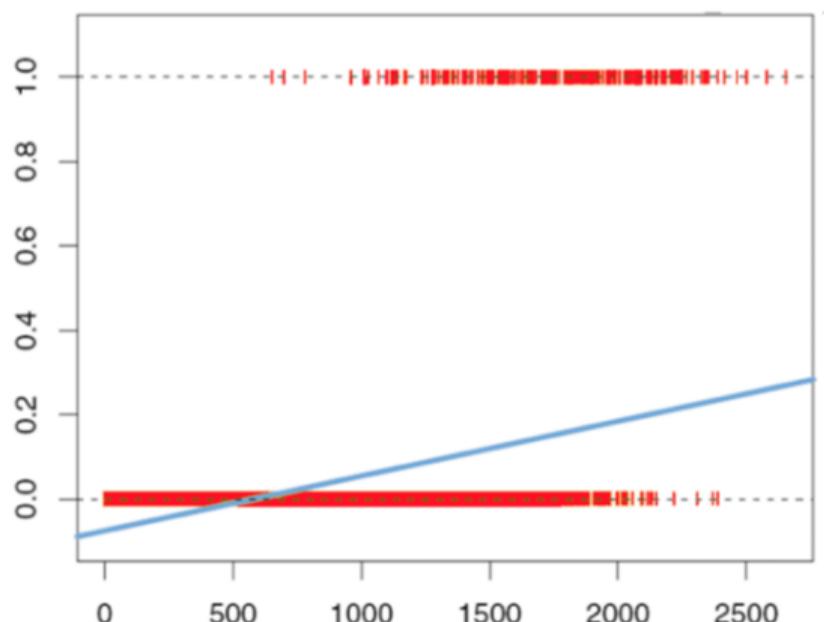
- $y \in \{0, 1\}$  — целевая переменная (болен или здоров, кот или собака и тп)
- $X$  — признаки
- Хотим оценить

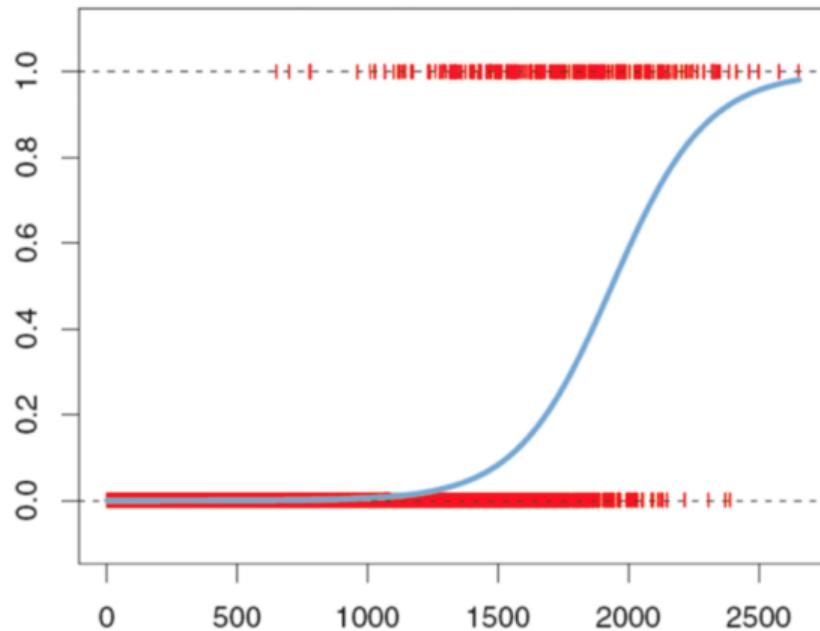
$$P(y = 1 \mid X) = w_0 + w_1x_1 + w_2x_2 + \dots + w_kx_k$$

- Нужно оценить  $k + 1$  параметр

## Предсказание вероятностей

- Мы хотим оценить вероятность, но  $w^T x_i$  не всегда лежит на отрезке  $[0; 1]$





# Предсказание вероятностей

- Возьмём функцию  $F$ , которая переводит  $w^T x_i$  на отрезок  $[0; 1]$

$$P(y = 1 \mid x) = F(w_0 + w_1 x_1 + w_2 x_2 + \dots + w_k x_k)$$

- Чаще всего в качестве  $F$  берут сигмоиду, логистическую функцию распределения

$$P(y = 1 \mid x) = F(w^T x) = \frac{e^{w^T x}}{1 + e^{w^T x}} = \frac{1}{1 + e^{-w^T x}}$$

- В её случае

$$\ln \frac{P(y = 1 \mid X)}{P(y = 0 \mid X)} = w^T x$$

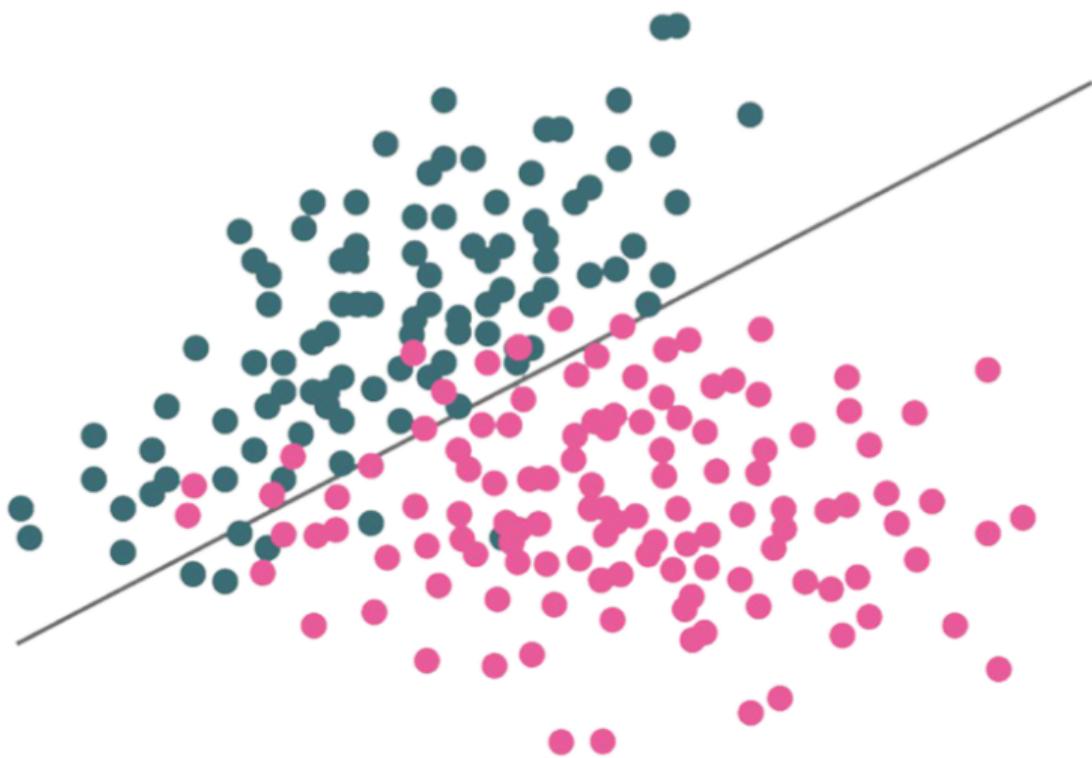
# Логистическая регрессия (функция потерь)

Как тренировать модель? Из метода максимального правдоподобия можно получить логистическую функцию потерь

$$L(w) = - \sum_{i=1}^n y_i \cdot \ln P(y_i = 1 \mid X) + (1 - y_i) \cdot \ln(1 - P(y_i = 1 \mid X))$$

Можно переписать

$$L(w) = - \sum_{i=1}^n \sum_{k \in \{0,1\}} [y_i = k] \cdot \ln \frac{e^{w^T x}}{1 + e^{w^T x}}$$



# Обобщение на много классов (softmax)

Много классов,  $y \in 1, \dots, K$

$$(w_1^T x, \dots, w_K^T x)$$



$$(e^{z_1}, \dots, e^{z_K})$$



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

Потери (кросс-энтропия):

$$L(w) = -\sum_{i=1}^n \sum_{k=1}^K [y_i = k] \cdot \ln \frac{e^{w_k^T x_i}}{\sum_{j=1}^K e^{w_j^T x_i}}$$

# Переобучения и регуляризация

## Оценка качества модели

- На тренировочной выборке модель показывает долю правильных ответов 80%
- Означает ли это, что на новых данных модель будет работать также хорошо?
- Обладает ли модель обобщающими способностями?

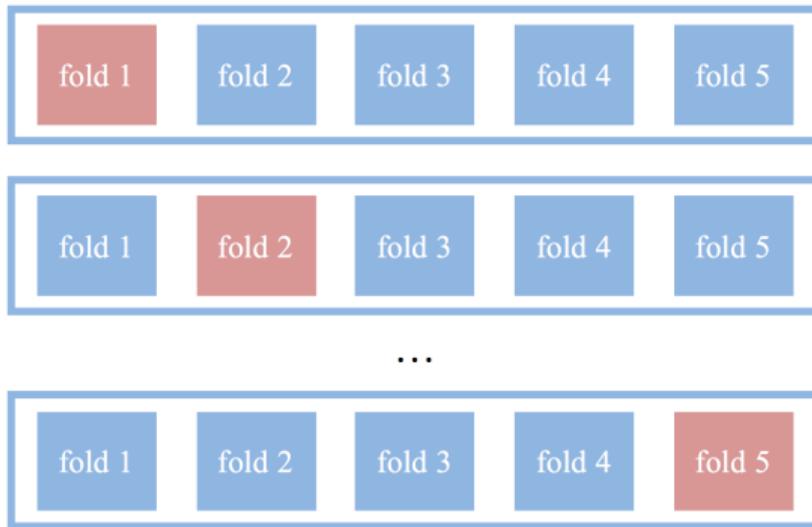
# Отложенная выборка

Training set

Holdout set

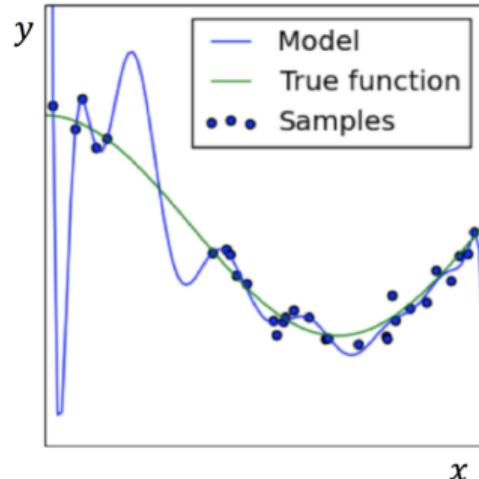
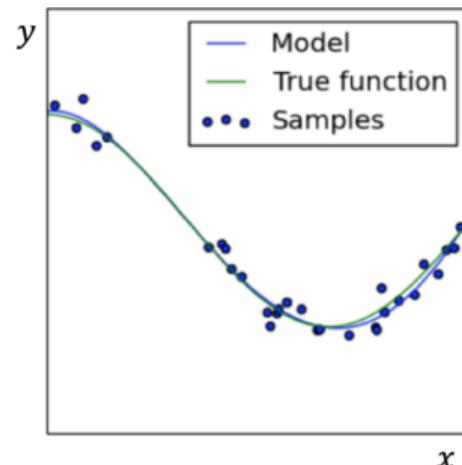
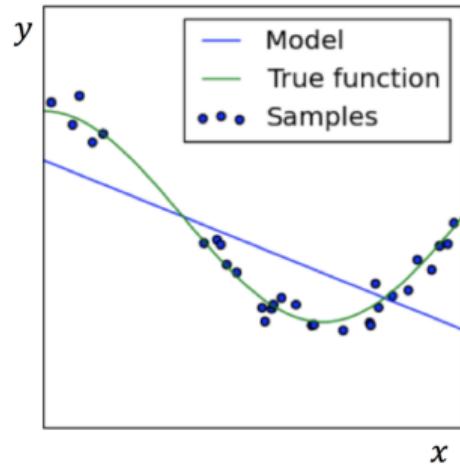
- На тренировочной выборке учим модель
- Для проверки качества используем отложенную (тестовую) выборку
- Итоговая оценка качества зависит от разбиения

# Кросс-валидация



- Нужно оценить  $K$  моделей, нет зависимости от разбиения
- Для нейросеток обычно не используют, потому что долго

# Переобучение



# Регуляризация

- В хорошей модели:  $(0.634, 0.918, -0.626)$
- В переобученной модели:  $(130.0, -525.8, \dots, 102.6)$
- Мысль: оштрафовать модель за излишне большие веса, это уберёт изгибы и сделает её проще

$$L(w) + \lambda \cdot R(w) \rightarrow \min_w$$

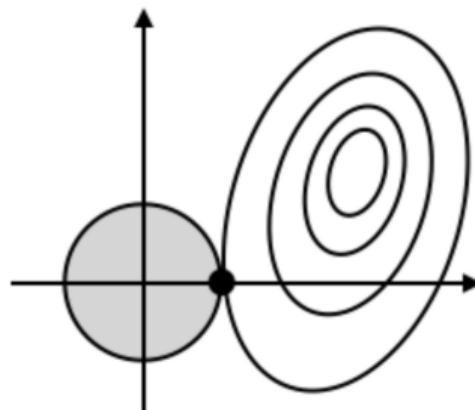
- В общем случае регуляризаторы помогают получить решение с желаемыми нам свойствами

## $l_2$ регуляризация

$$L(w) + \lambda \cdot \sum w_i^2 \rightarrow \min_w$$

Задача оптимизации эквивалентна:

$$\begin{cases} L(w) \rightarrow \min_w \\ \sum w_i^2 \leq C \end{cases}$$

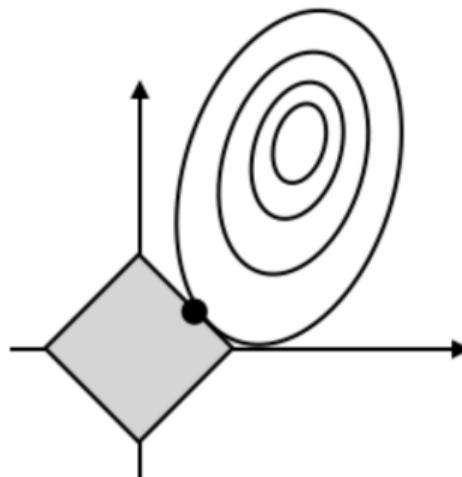


## $l_1$ регуляризация

$$L(w) + \lambda \cdot \sum |w_i| \rightarrow \min_w$$

Задача оптимизации эквивалентна:

$$\begin{cases} L(w) \rightarrow \min_w \\ \sum |w_i| \leq C \end{cases}$$



- Регуляризация помогает избежать переобучения, ситуации, когда модель излишне подробно вылизывает данные и настраивается на шум, вместо выделения сигнала
- Нейросетки довольно сложные модели, включающие огромное количество параметров, на маленькой выборке их довольно легко переобучить
- Для борьбы с переобучением мы будем использовать как уже знакомые нам регуляризаторы, так и новые техники

# Градиентный спуск

# Как обучать?

- «Тренировка» — поиск оптимальных  $W$
- «Оптимальных» — минимизирующих какой-то функционал
- Какими бывают функционалы: MSE, MAE, Logloss
- Как оптимизировать: градиентный спуск!

# Градиентный спуск (GD)

Проблема оптимизации:

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

Инициализация  $w_0$

**while** True:

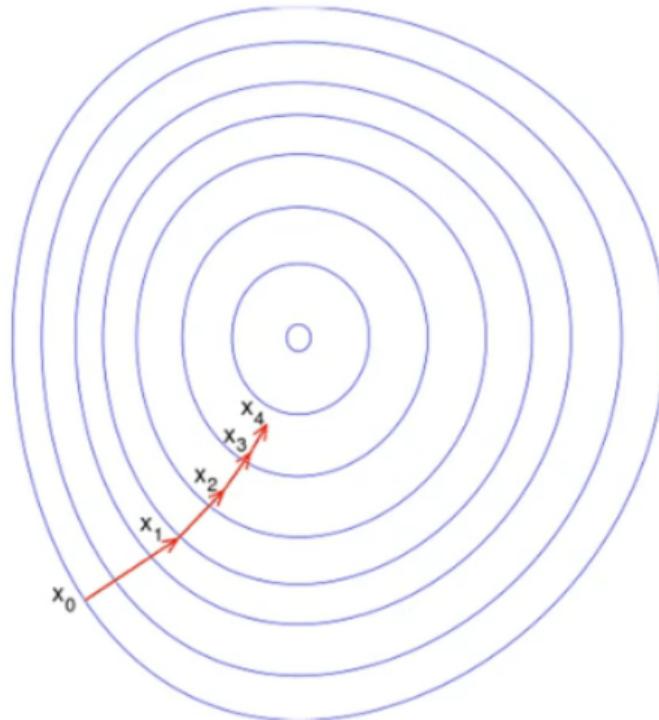
$$g_t = \frac{1}{n} \sum_{i=1}^n \nabla L(w, x_i, y_i)$$

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

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

**break**

# Градиентный спуск



# Стochastic gradient descent (SGD)

Проблема оптимизации:

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

Инициализация  $w_0$

**while** True:

рандомно выбрали  $i$

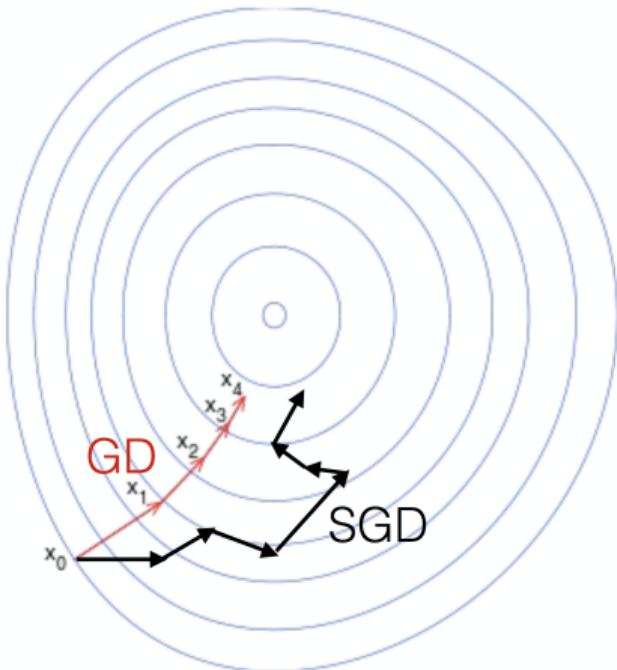
$$g_t = \nabla L(w_{t-1}, x_i, y_i)$$

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

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

**break**

# 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:  $O(1)$
- Скорость обучения надо подбирать аккуратно, если она будет большой, мы можем скакать вокруг минимума, если маленькой - вечно ползти к нему

# Mini-bath SGD

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**

# Momentum SGD

Мы считали на каждом шаге градиент по формуле

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

После шага мы забывали его. **А давайте попробуем помнить направление:**

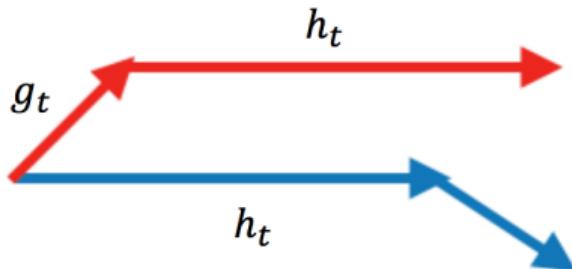
$$\begin{aligned} h_t &= \alpha \cdot h_{t-1} + \eta_t \cdot g_t \\ w_t &= w_{t-1} - h_t \end{aligned}$$

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

# 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 смотреть себе под ноги, чтобы внезапно не споткнуться  $\Rightarrow$  давайте смотреть на градиент в будущей точке
- Согласно методу моментов  $\alpha h_{t-1}$  точно будет использоваться при шаге, давайте искать  $\nabla L(w_{t-1} - \alpha \cdot h_{t-1})$ .

# Nesterov Momentum SGD



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

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

# Momentum SGD

- 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

# AdaGrad

$$\begin{aligned}G_t^j &= G_{t-1}^j + g_{tj}^2 \\w_t^j &= w_{t-1}^j - \frac{\eta_t}{\sqrt{G_t^j + \varepsilon}} \cdot g_t^j\end{aligned}$$

- $g_t^j$  — градиент по  $j$ -ому параметру
- своя скорость обучения для каждого параметра
- обычно  $\eta_t = 0.01$ , этот параметр не имеет значения
- $G_t^j$  всегда увеличивается, из-за этого обучение может рано останавливаться

# 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

# Adam

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

Комбинируем momentum и индивидуальные скорости обучения

## Подведём итоги

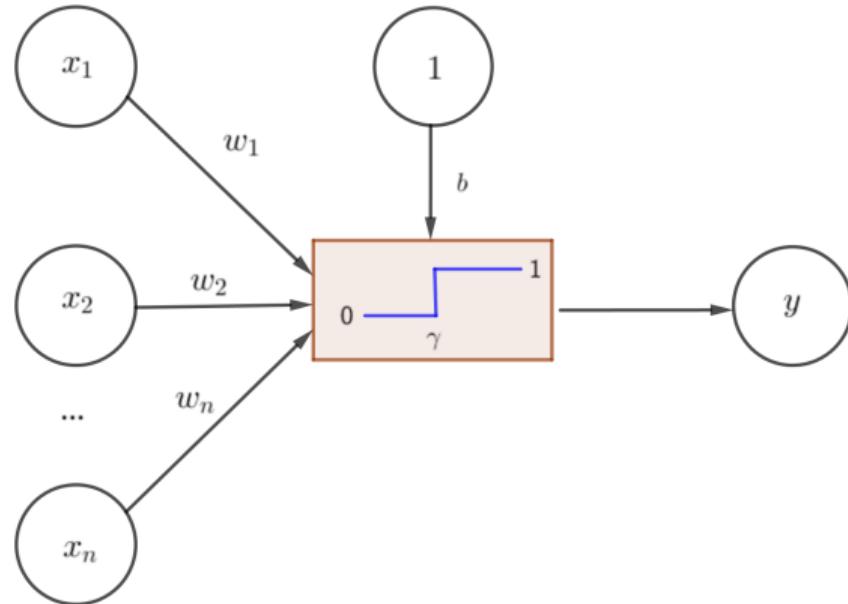
- Momentum SGD сохраняет направление шага и позволяет добиваться более быстрой сходимости
- Адаптивные методы позволяют находить индивидуальную скорость обучения для каждого параметра
- Adam комбинирует в себе оба подхода

# Перцепtron

# Перцептрон

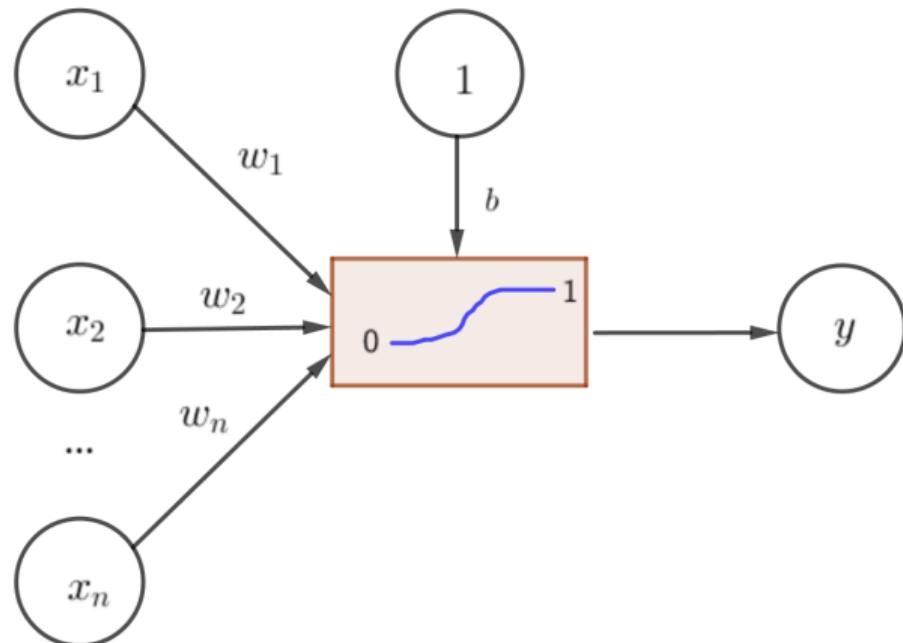
- Перцептрон — древняя штука, придуманная Розенблатом в 1950-е годы.

# Перцептрон Розенблатта

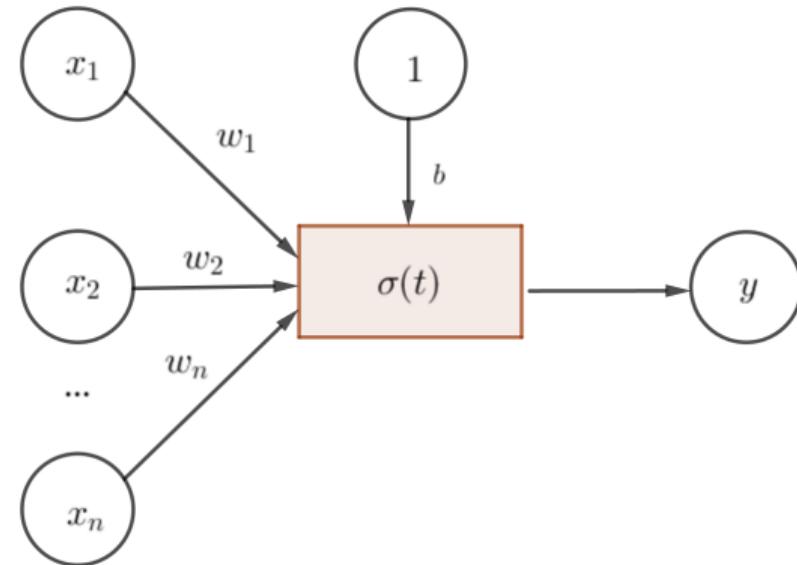


$$y = \begin{cases} 1, & \text{если } \sum w_i x_i \geq \gamma \\ 0, & \text{если } \sum w_i x_i < \gamma \end{cases}$$

# Немного иной перцептрон Розенблатта



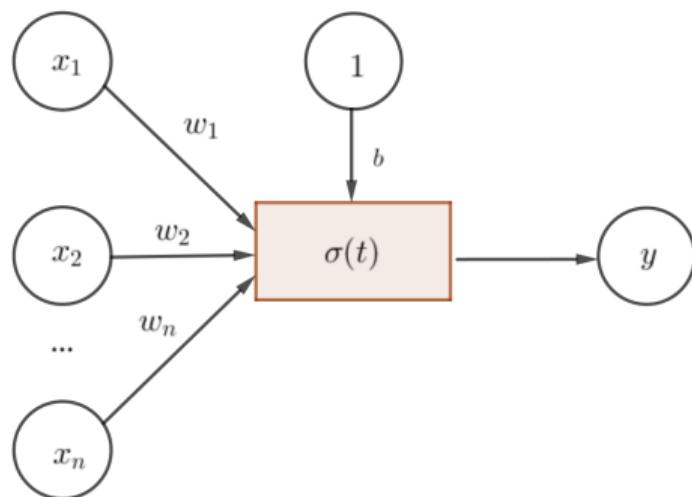
# Функция активации



- Функция активации  $\sigma(t)$  вносит нелинейность, она может быть любой

# Линейная регрессия

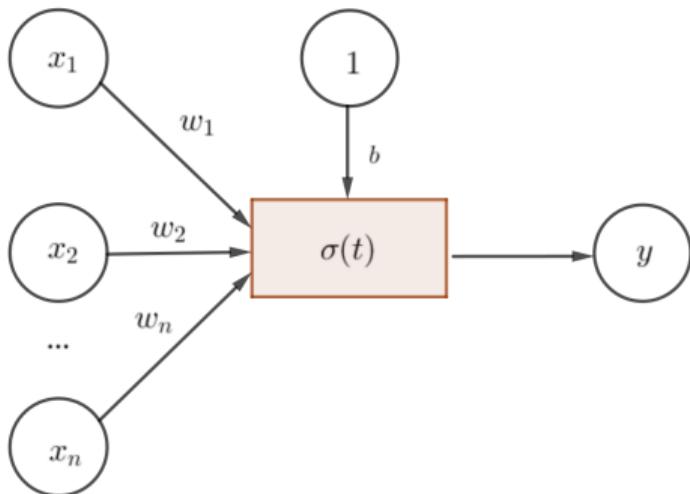
Нейрон с линейной функции  
активации — это линейная регрессия...



$$\sigma(t) = t$$

$$y = b + w_1x_1 + \dots + w_nx_n$$

# Логистическая регрессия

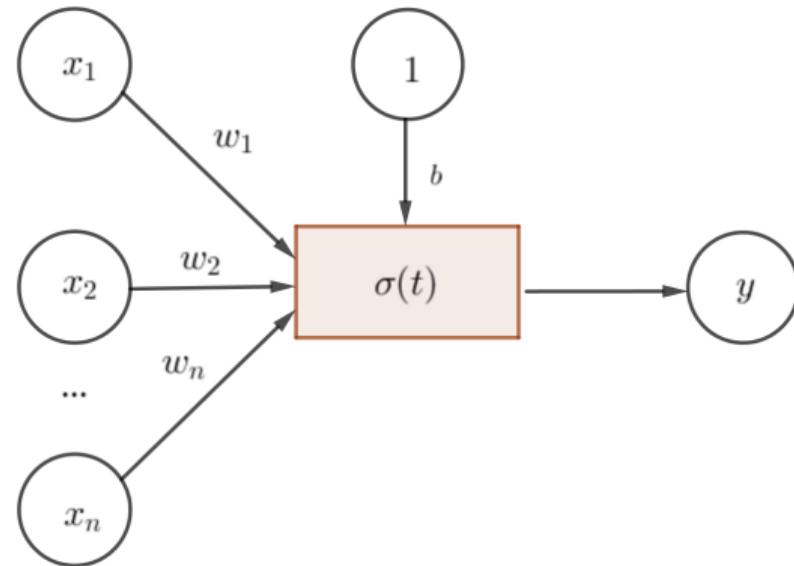


Нейрон с сигмоидом в качестве функции активации — это логистическая регрессия...

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

$$P(y = 1 | x) = \sigma(b + w_1x_1 + \dots + w_nx_n)$$

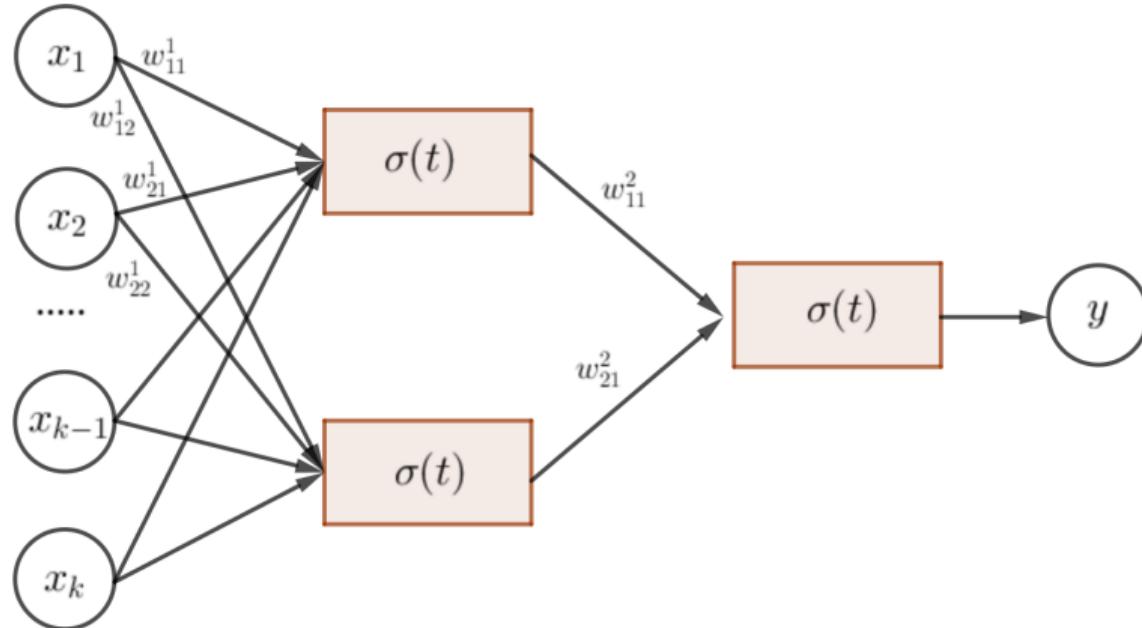
# Функция активации



$$y = \sigma(w_0 + w_1 \cdot x_1 + \dots + w_n \cdot x_n) = \sigma(Xw)$$

$$h = Xw$$

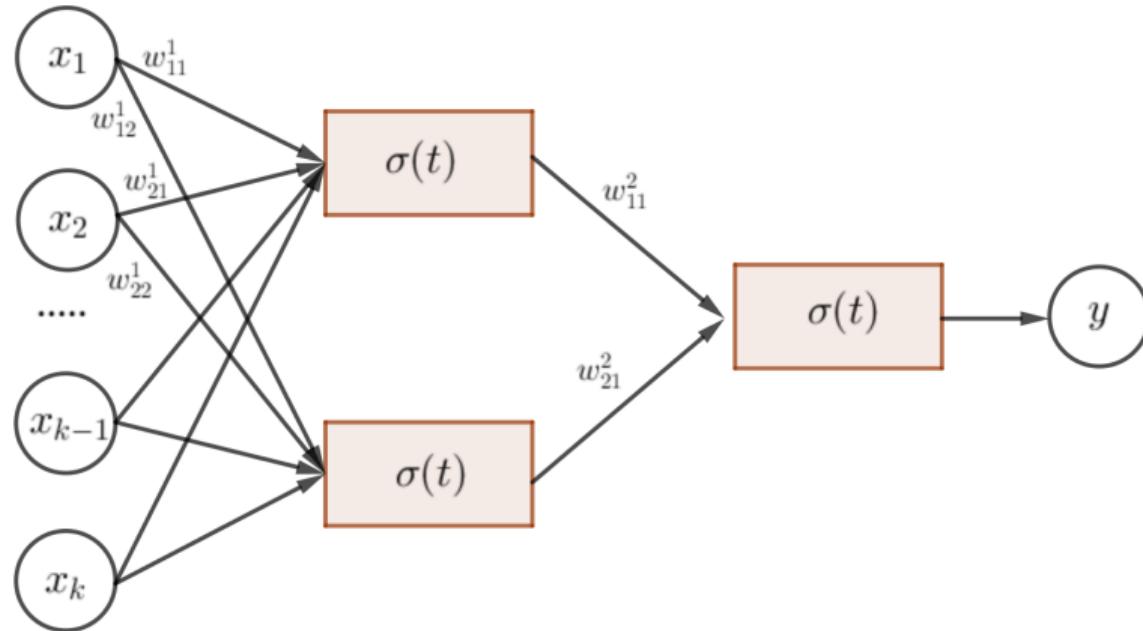
# Две регрессии скрепили третьей



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

$$h_j = \sigma\left(\sum_i w_{ij}\right)$$

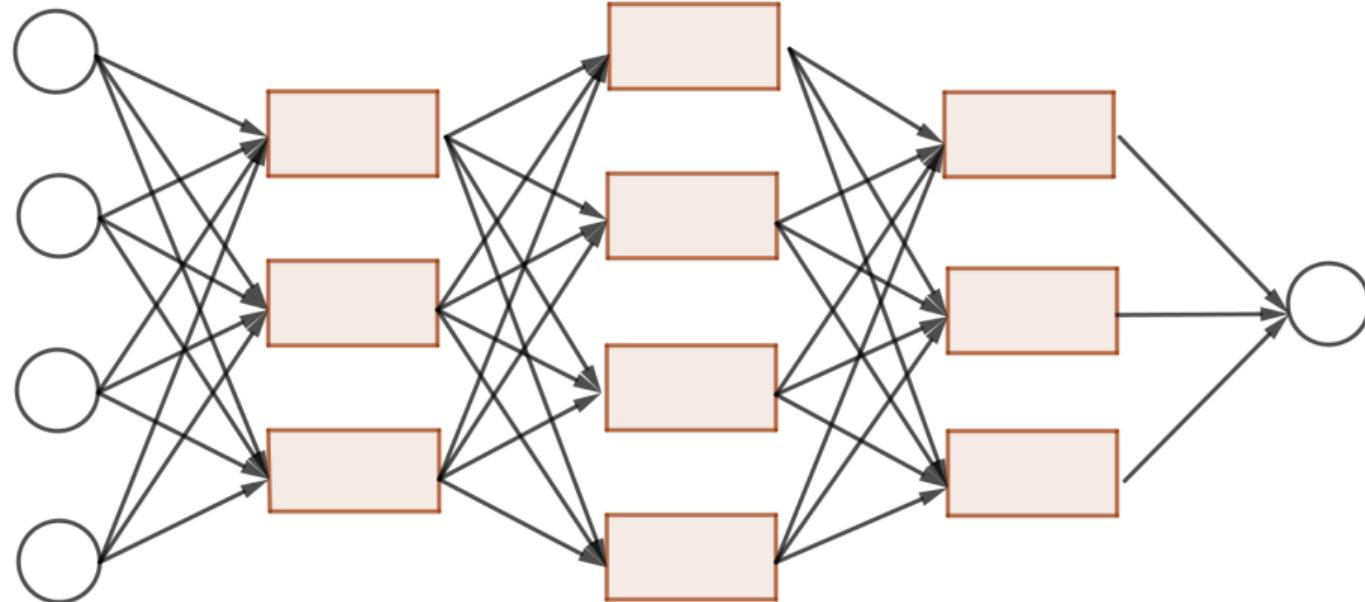
# Две регрессии скрепили третьей



$$y = \sigma(hW)$$

$$h = \sigma(XW)$$

# Армия из регрессий



# The Perceptron Convergence Theorem (Rosenblat, 1965)

- Любая непрерывная и ограниченная функция может быть сколь угодно точно аппроксимирована нейронной сетью с одним скрытым слоем с нелинейной функцией активации нейрона.
- Любая функция может быть сколь угодно точно аппроксимирована нейронной сетью с двумя скрытыми слоями с нелинейной функцией активации нейрона.
- Что ещё можно пожелать?

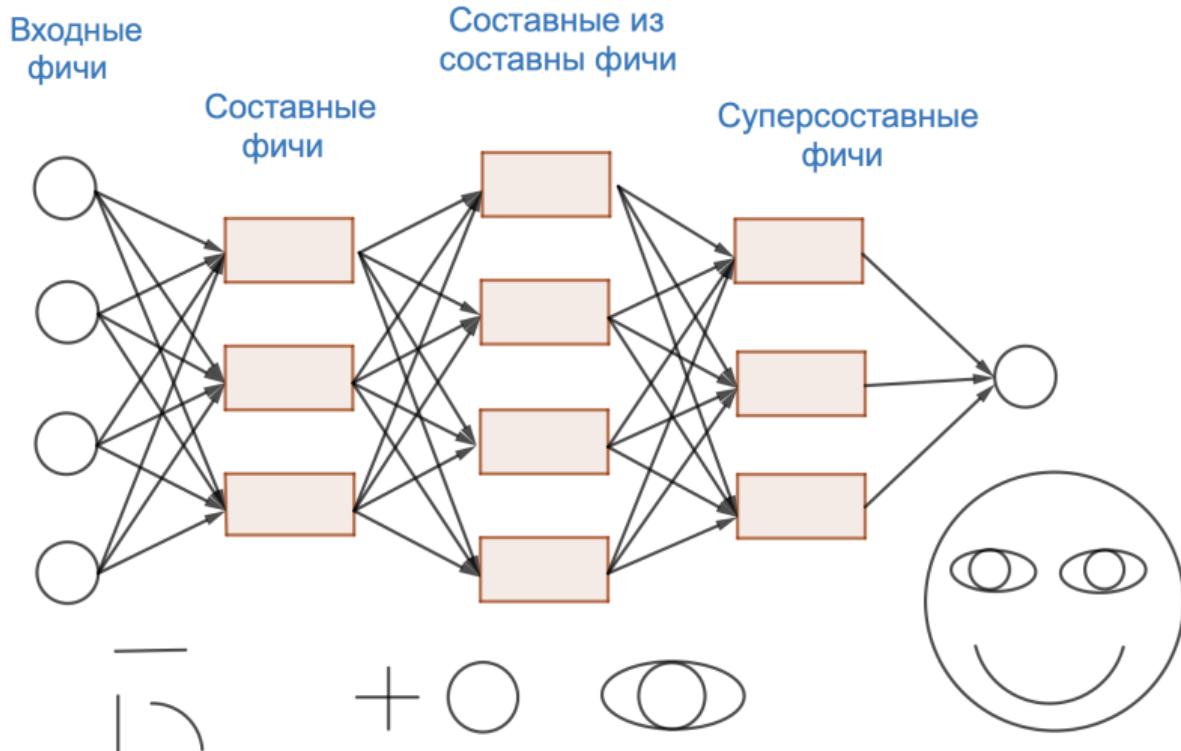
A photograph taken from underwater looking up at the surface. The water is a deep, vibrant blue. Sunlight filters down from the surface in bright, dappled rays and patches, creating a play of light and shadow on the water's surface. The overall atmosphere is serene and suggests depth.

Going Deeper

# Мотивация

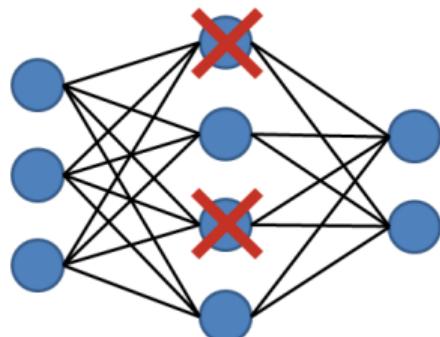
- Перцептрон может решить любую проблему, но это дорого
- Глубокие архитектуры часто позволяют выразить то же самое, приблизить те же функции гораздо более эффективно, чем неглубокие
- Каждый новый слой сетки будет работать всё с более сложными фичами

# Армия из регрессий



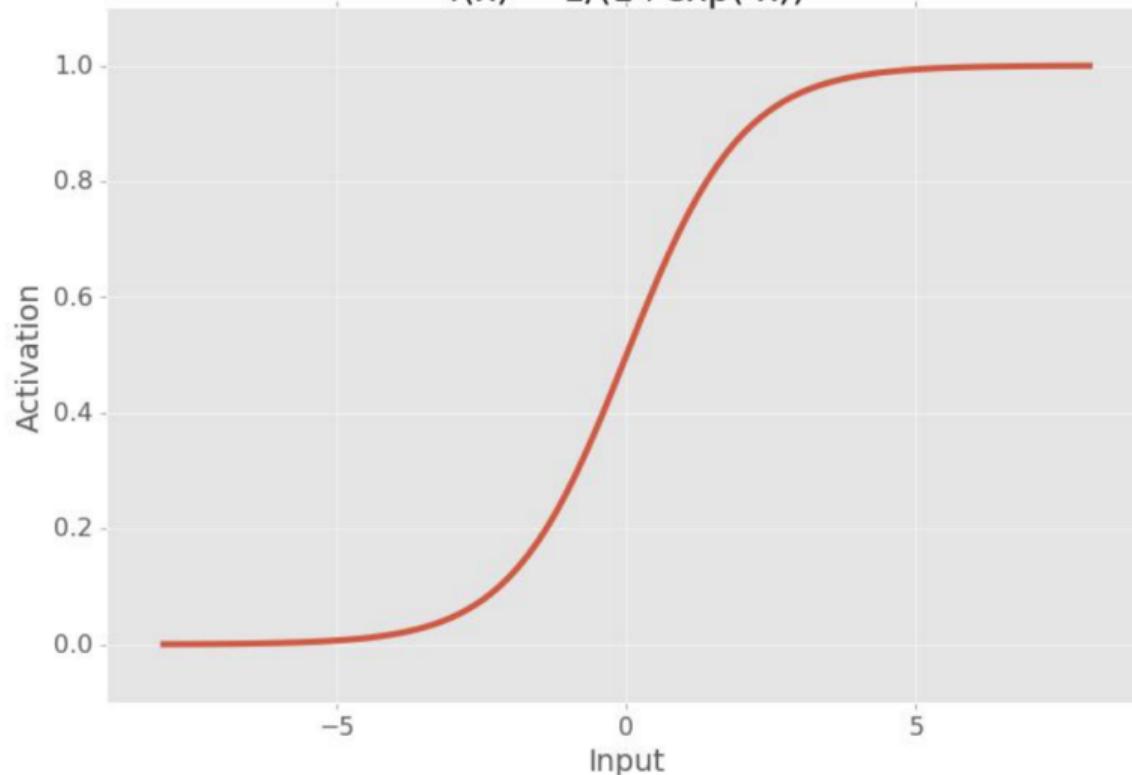
## Слои бывают разными

- Слой, который просто взвешивает входы называется **полносвязным**.
- Слои бывают очень разными. Например, **Dropout**: с вероятностью  $p$  отключаем нейрон. Такой слой препятствует переобучению и делает нейроны более устойчивыми к случайным возмущениям.

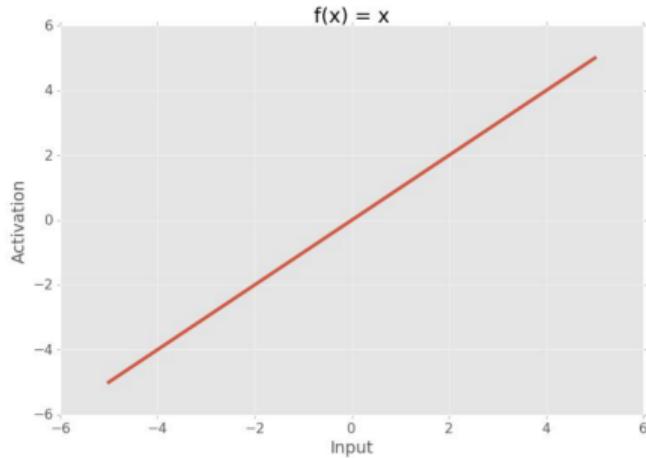


## Функции активации бывают разными (сигмоид)

$$f(x) = 1/(1+\exp(-x))$$



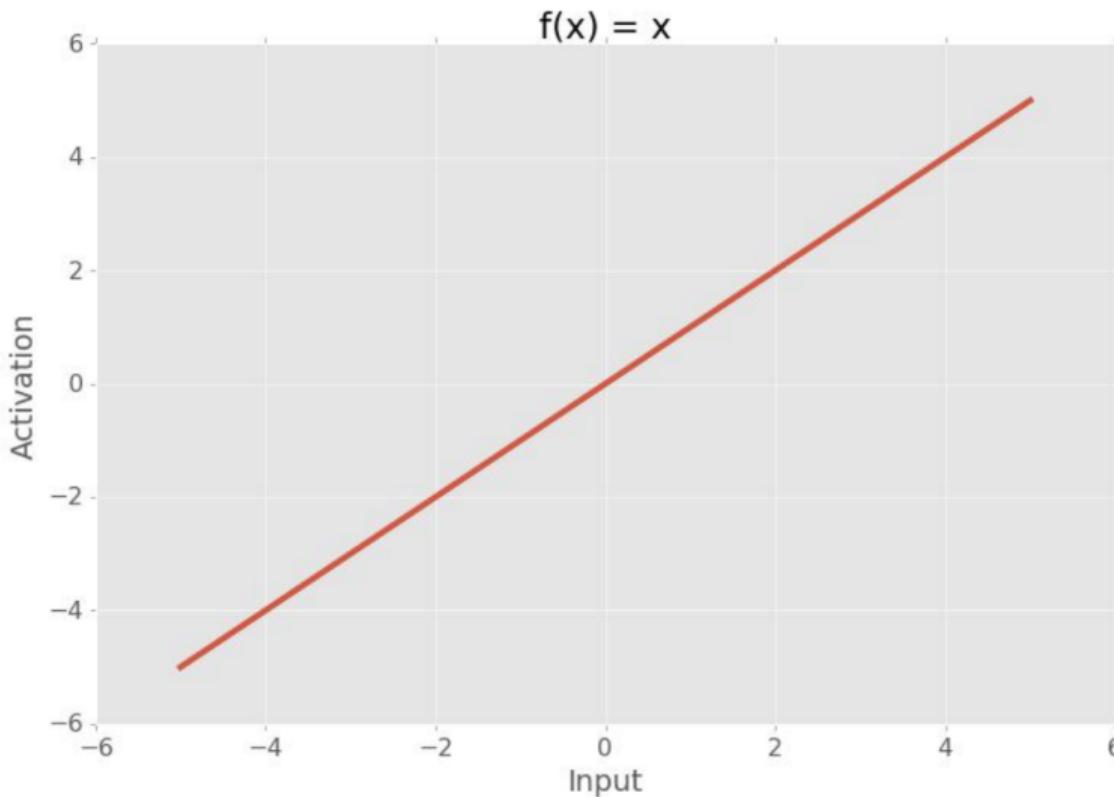
# Линейная активация



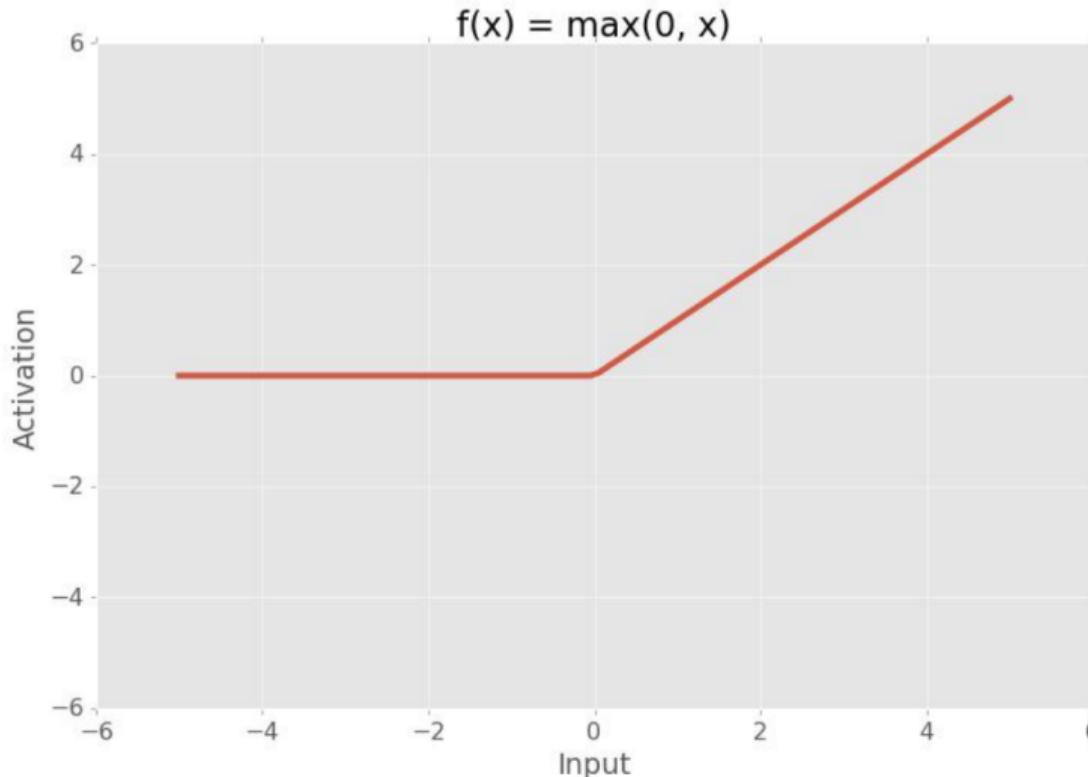
$$\hat{y} = h \cdot W_2 = X \cdot W_1 \cdot W_2 = X \cdot A$$

- После линейной активации выход снова линейный

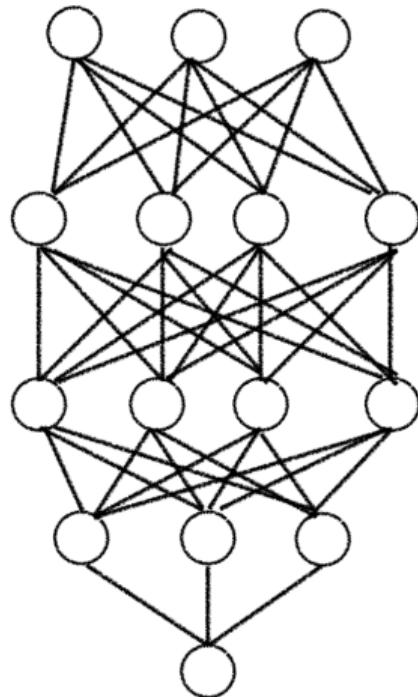
# От линейной активации ...



## ... к нелинейной (ReLU)



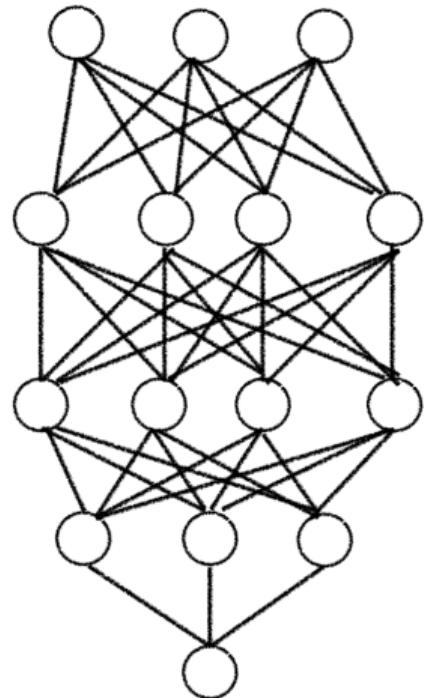
# Архитектуры бывают разными



Input	
Fully connected layer (FC)	$XW + b$
ReLU	$\max(0, x)$
Dropout	$Bern(p)$
FC	$XW + b$
ReLU	$\max(0, x)$
FC	$XW + b$
Output	

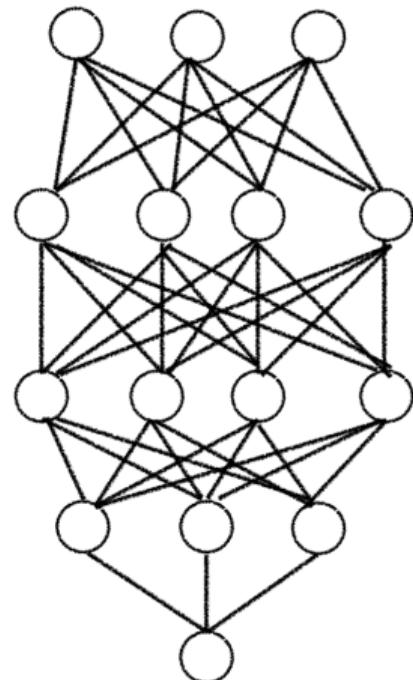
Каждый слой — просто функция, каждая сетка — конструктор LEGO

# Регрессия



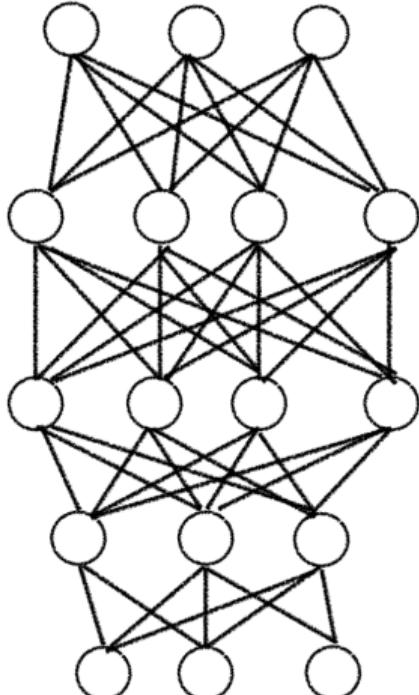
Input	
Fully connected layer (FC)	$XW + b$
ReLU	$\max(0, x)$
Dropout	$Bern(p)$
FC	$XW + b$
ReLU	$\max(0, x)$
FC	$XW + b$
Output	

# Классификация



<b>Input</b>	
<b>Fully connected layer (FC)</b>	$XW + b$
<b>ReLU</b>	$\max(0, x)$
<b>Dropout</b>	$Bern(p)$
<b>FC</b>	$XW + b$
<b>ReLU</b>	$\max(0, x)$
<b>FC</b>	$XW + b$
<b>Sigmoid</b>	$\sigma(x)$
<b>Output</b>	

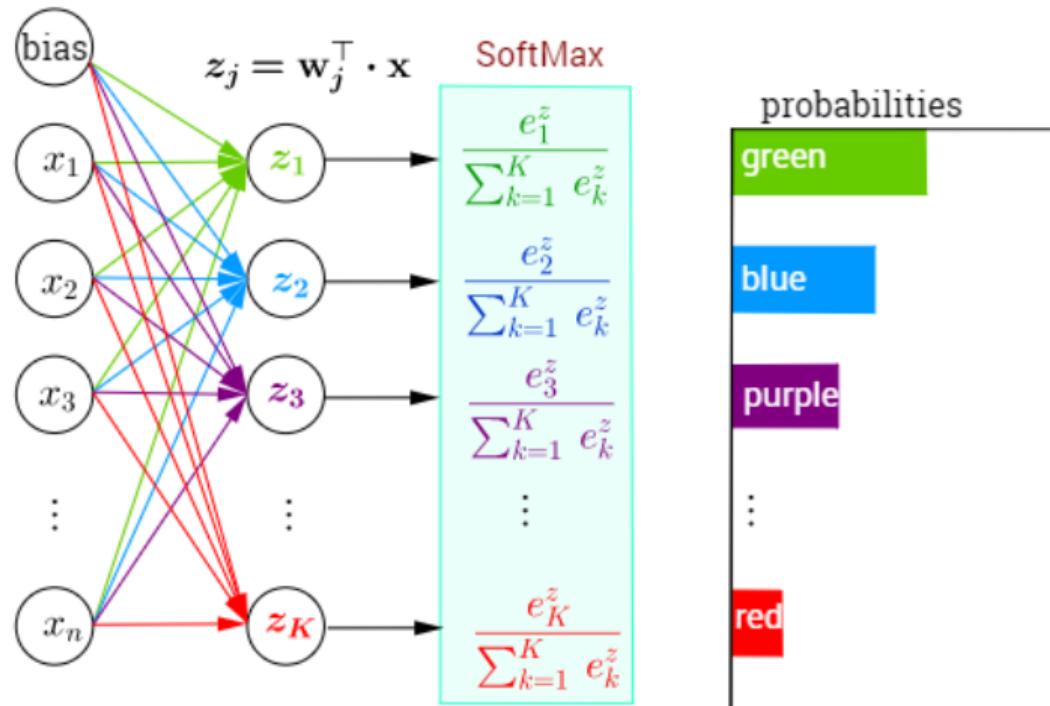
# Мультиклассификация



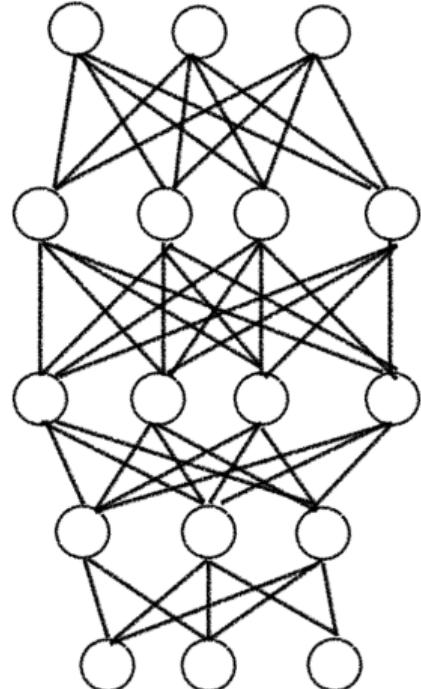
Input	
Fully connected layer (FC)	$XW + b$
ReLU	$\max(0, x)$
Dropout	$Bern(p)$
FC	$XW + b$
ReLU	$\max(0, x)$
FC	$XW + b$
Softmax	$\text{softmax}(x)$
Output	

# Мультиклассификация

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$



# Мультирегрессия



Input	
Fully connected layer (FC)	$XW + b$
ReLU	$\max(0, x)$
Dropout	$Bern(p)$
FC	$XW + b$
ReLU	$\max(0, x)$
FC	$XW + b$
Output	

Учим нейросеть руками!