Глубинное обучение

Коломейцева Катерина

Лекция 1: вводная

2023

Agenda

- О том каким будет курс + что почитать/посмотреть
- Фреймворки для нейроночек
- Немного истории и важные тренды
- От регрессии к нейросетке
- Нейросетки - конструктор Lego
- Учим свою первую нейросетку

Правила игры и ссылки на доп. ресурсы

Занятия

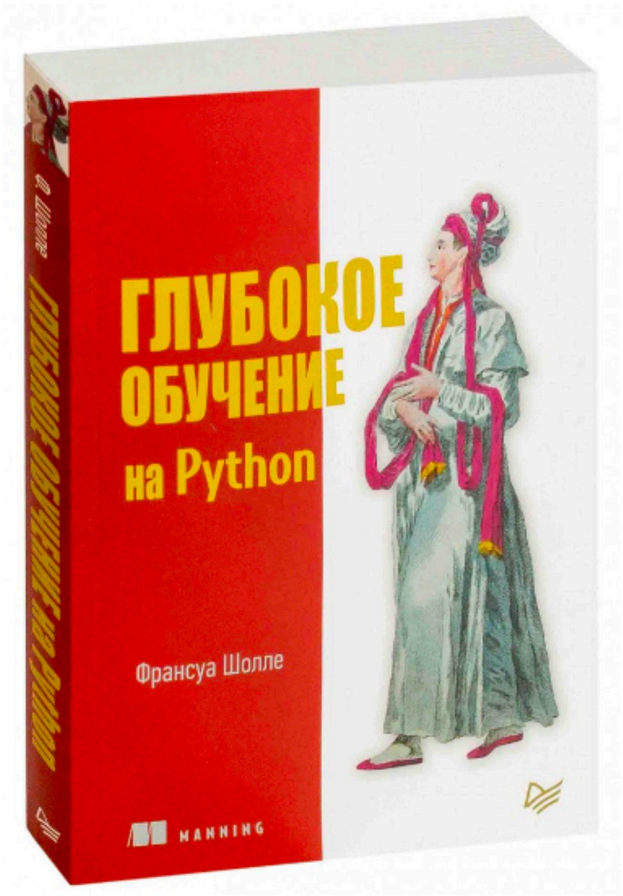
- Первая часть занятия - лекция, вторая - семинар.
- На парах смотрим презентации, пишем код, дискутируем - будет определенное кол-во математики
- Все материалы можно найти на страничке курса
- Домашние задания

Что почитать про нейронки в первую очередь

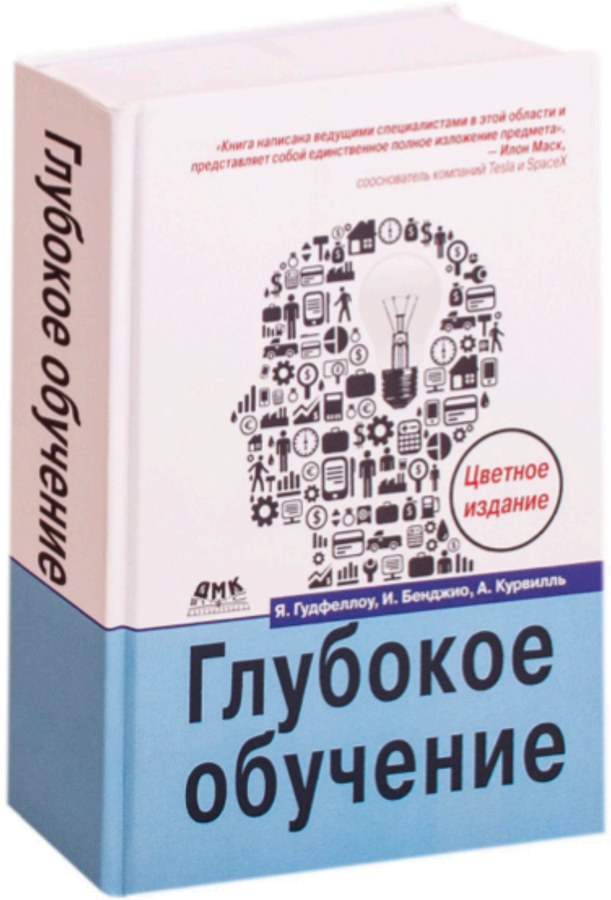
Баланс математики и практики,
код устарел (tensorflow 1.14)

Философия и история ML

Что почитать про нейронки в первую очередь



Простая практика на Keras



Хардкорная математика,
библия глубокого обучения

Что посмотреть про нейронки

- Если ничего не знаете про машинное обучение, смотрите вводный курс от Яндекса и МФТИ. Для тех, кто мало знает про ML.

- Advanced ML от Яндекса. Там есть очень разные специфические курсы. Первый из них про нейронки. Код на Tensorflow. Версия библиотеки там пока что старая. Для тех, кто хочет развиваться дальше.

- Нейронные сети от Andrew Ng. Когда ML ещё не был таким модным, все смотрели его лекции. Для тех, кто хочет всё делать медленно и непринуждённо.

- Курс нейронок, который читают в ШАД и Сколтехе. Есть варианты кода на разных фреймворках. Есть видео лекций на русском и английском. Для тех, кто хочет посмотреть как читают курс по DL в ШАД.

Что посмотреть про нейронки

- Бесплатный курс по tf от Google. Короткий. Покрывает весь базовый Keras. Все тетрадки выложены в colab. Есть странные интервью. Для тех, кто хочет быстро разобраться в keras.

- Бесплатный курс по pytorch от Samsung. Вводный курс в нейросетки на Pytorch. Для тех, кто хочет писать на pytorch.

- Deep learning на пальцах. Лекции из кремниевой долины. Задания на pytorch для самостоятельного решения. Для тех, кто хочет посмотреть простые рассказы про нейронки.

Структура курса

1. Введение в глубокое обучение. От линейной регрессии к нейронной сети. Смотрим на базовые возможности pytorch, собираем первую нейросеть.
2. Обучение нейронных сетей. Алгоритм обратного распространения ошибки.
3. Оптимизация. Эвристики для обучения нейронных сетей. Нормализация по батчам, инициализация и др.
4. Свёрточные нейронные сети. Классификация изображений.
5. Архитектуры свёрточных сетей. Что видят свёрточные сети. Transfer learning.
6. Обзор задач компьютерного зрения. Детекция, сегментация, перенос стиля, распознавание лиц.
7. Автокодировщики и генеративные модели (Generative Adversarial Networks).
8. Глубокое обучение для NLP. Векторные представления текстов: word2vec, fasttext.
9. Рекуррентные нейронные сети, работа с последовательностями. ELMO-эммбеддинги.
10. Sequence2sequence, архитектуры с механизмом внимания. Трансформеры. BERT.

Фреймворки для Deep Learning

Фрэймворки

theano

Mila

Монреальский университет (2007)

TensorFlow

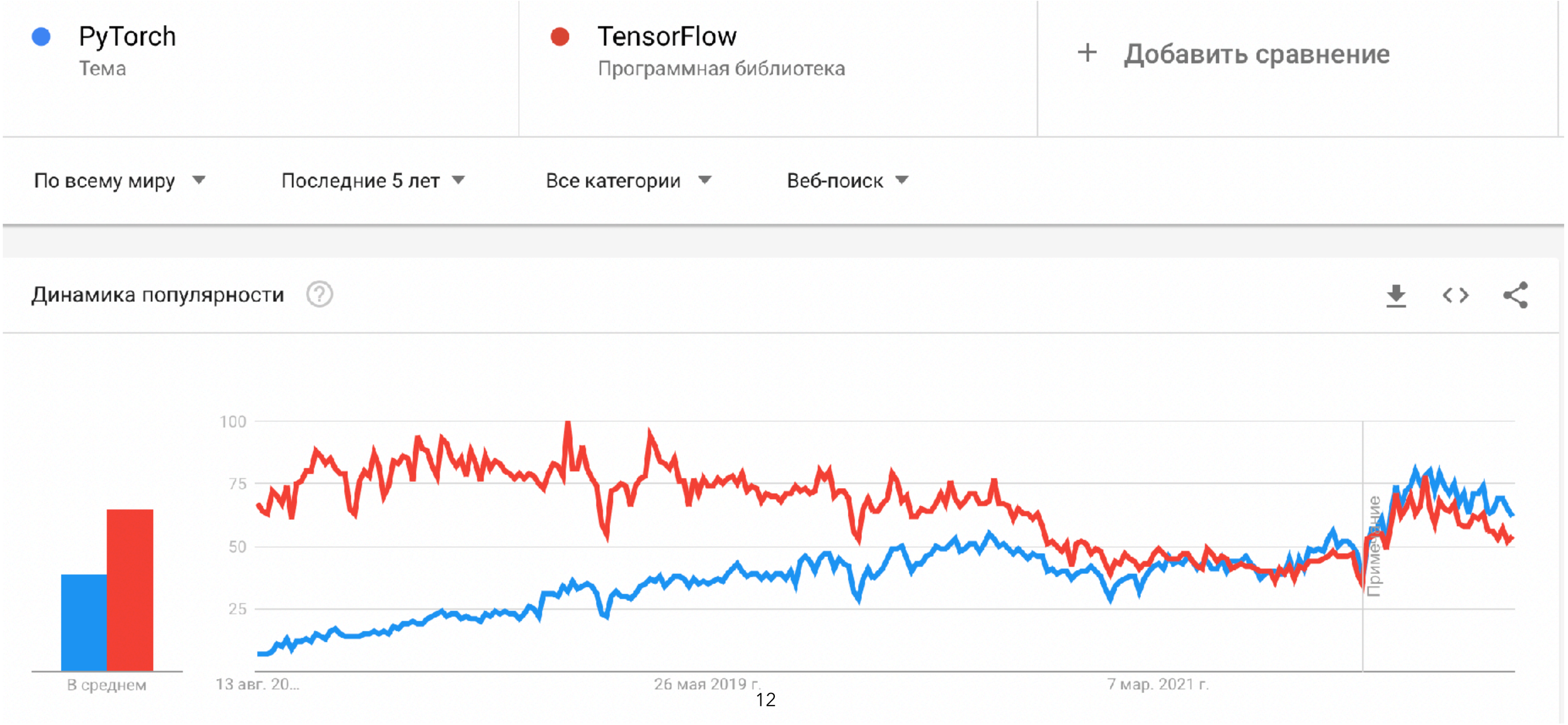
Google

Google (2011, открыта с 2015, с 2019 tf 2.0)

PYTORCH

Facebook (2016)

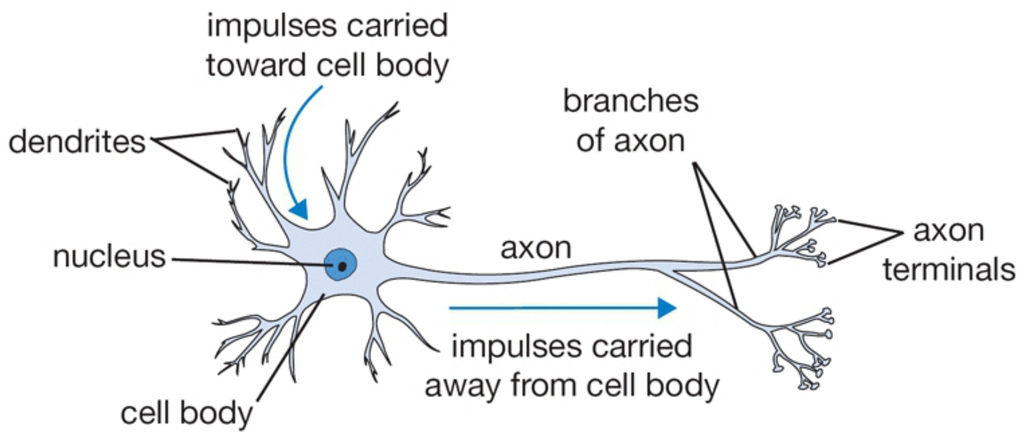
Google Trends

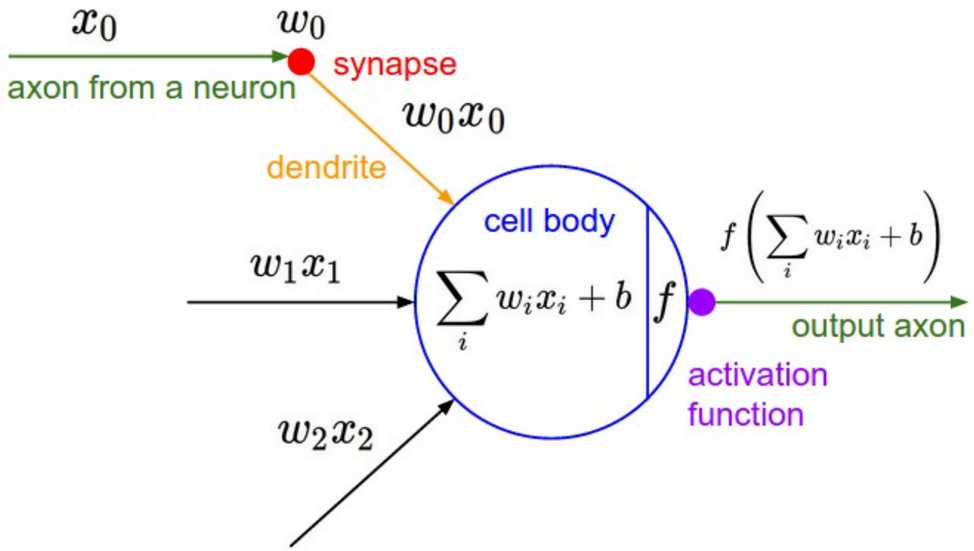


Немного истории

Biological view

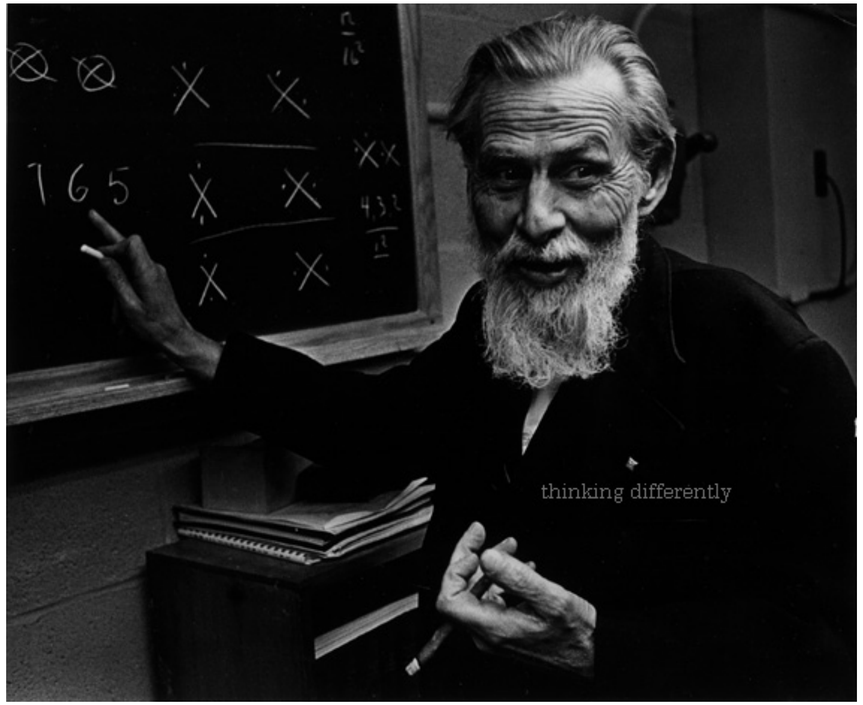
- Структура оказывается похожа на вполне биологический нейрон
- Более того, даже активация почти та же!
- Однако, биологический нейрон обучается как-то по-другому…



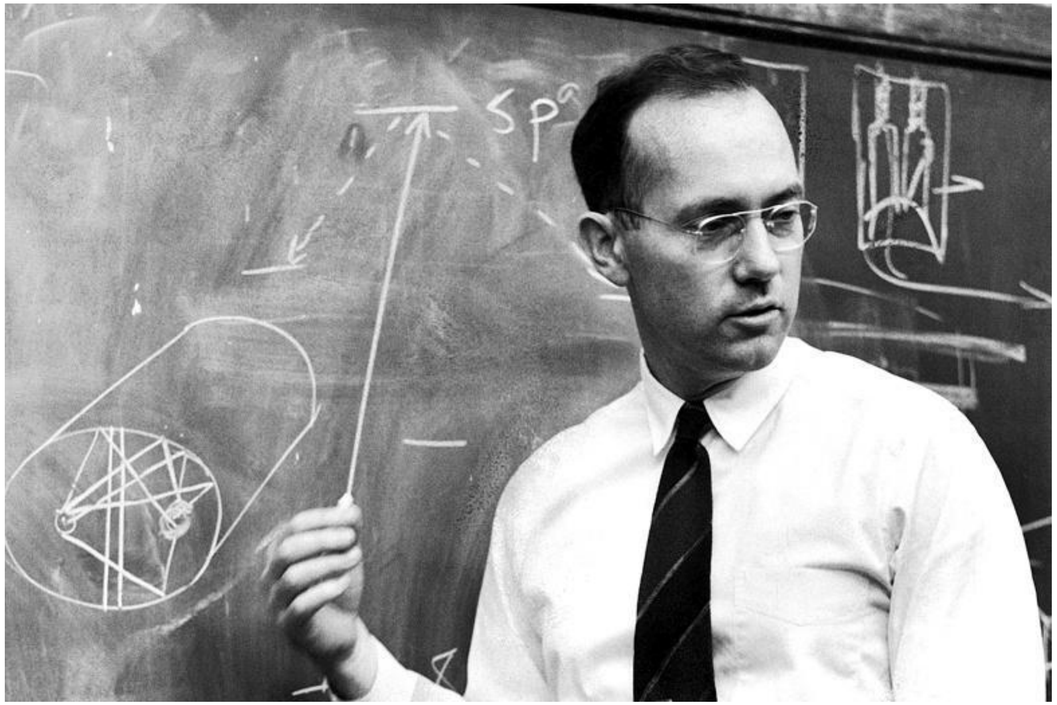


Historical view

- 1943 — Маккалок и Питтс формализуют понятие нейрона



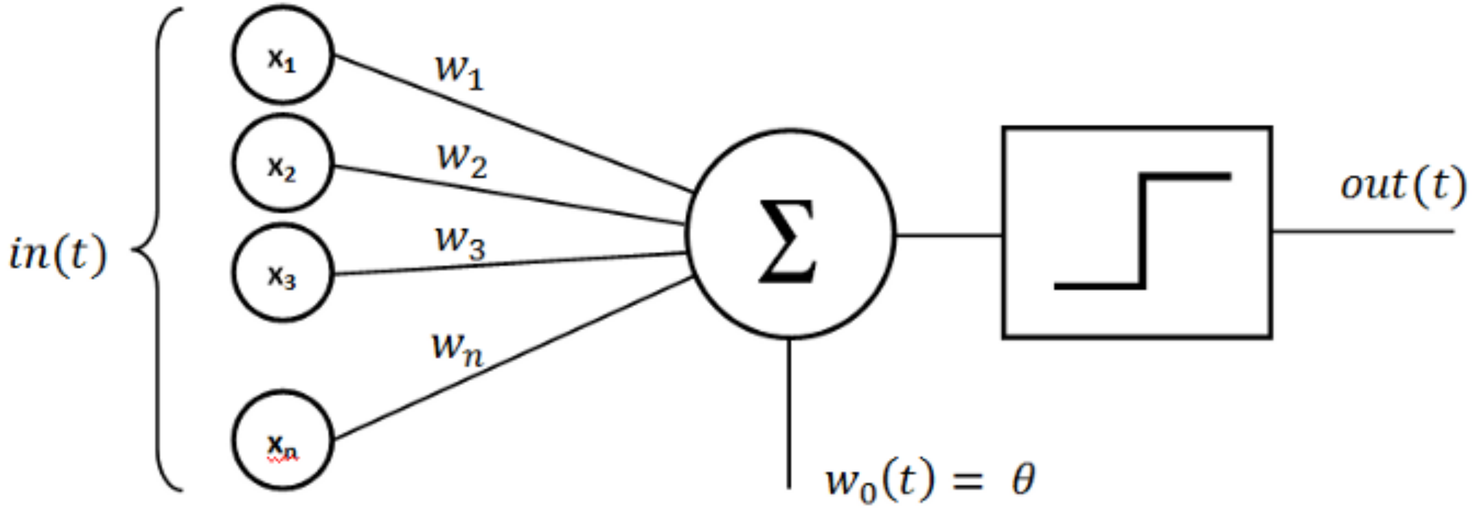
Уоррен Маккалок



Уолтер Питтс

Historical view

- 1943 — Маккалок и Питтс формализуют понятие нейрона
- 1949 — Хебб предлагает первый алгоритм обучения перцептрона
- 1958 — Розенблатт изобретает однослойный персептрон и демонстрирует его способность решать задачу классификации



Historical view

- 1943 — Маккалок и Питтс формализуют понятие нейрона
- 1949 — Хебб предлагает первый алгоритм обучения перцептрона
- 1958 — Розенблатт изобретает однослойный персептрон и демонстрирует его способность решать задачу классификации
- 1974 — Галушкин и Вербос предлагают метод обратного распространения ошибки (впоследствии улучшен Хинтоном в 1986)
- 1975–1980 — Фукусима предлагает мощную модель когнитрона и неокогнитрона (прототип сверточных сетей)
- 1989 — Лекун представляет первую работу по объединению свёрточных нейросетей и метода обратного распространения ошибки и добивается хорошей точности на датасете MNIST

Lecun’s Demo from 1993
https://www.youtube.com/watch?v=FwFduRA_L6Q

Historical view

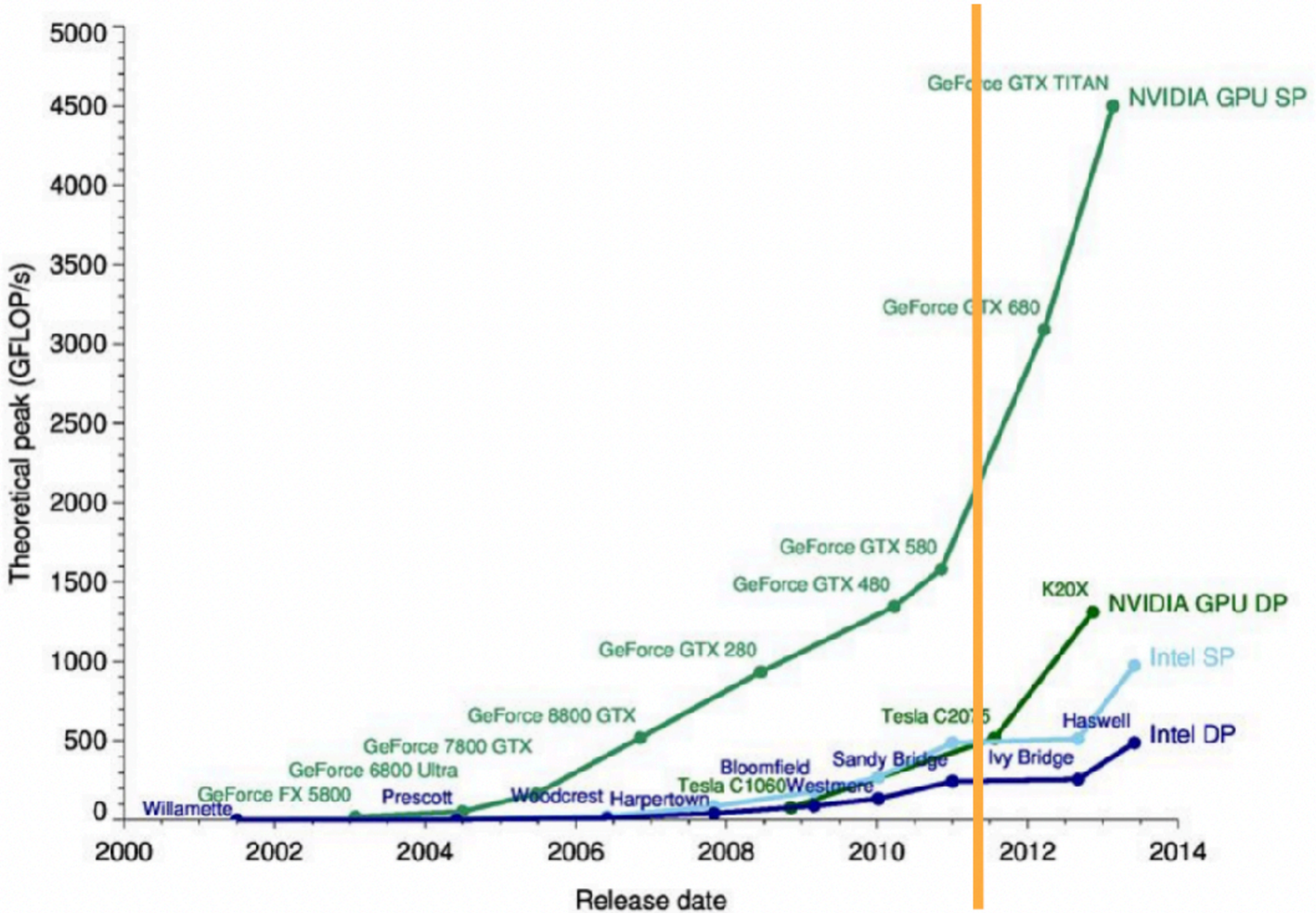
- 1943 — Маккалок и Питтс формализуют понятие нейрона
- 1949 — Хебб предлагает первый алгоритм обучения перцептрона
- 1958 — Розенблатт изобретает однослойный персептрон и демонстрирует его способность решать задачу классификации
- 1974 — Галушкин и Вербос предлагают метод обратного распространения ошибки (впоследствии улучшен Хинтоном в 1986)
- 1975–1980 — Фукусима предлагает мощную модель когнитрона и неокогнитрона (прототип сверточных сетей)
- 1989 — Лекун представляет первую работу по объединению свёрточных нейросетей и метода обратного распространения ошибки и добивается хорошей точности на датасете MNIST

Почему же нейросети не стали сразу популярны в 1990м году??

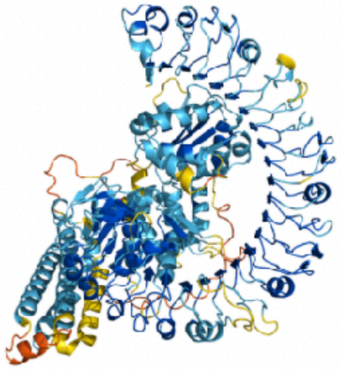
Чего же все ждали 20 лет?

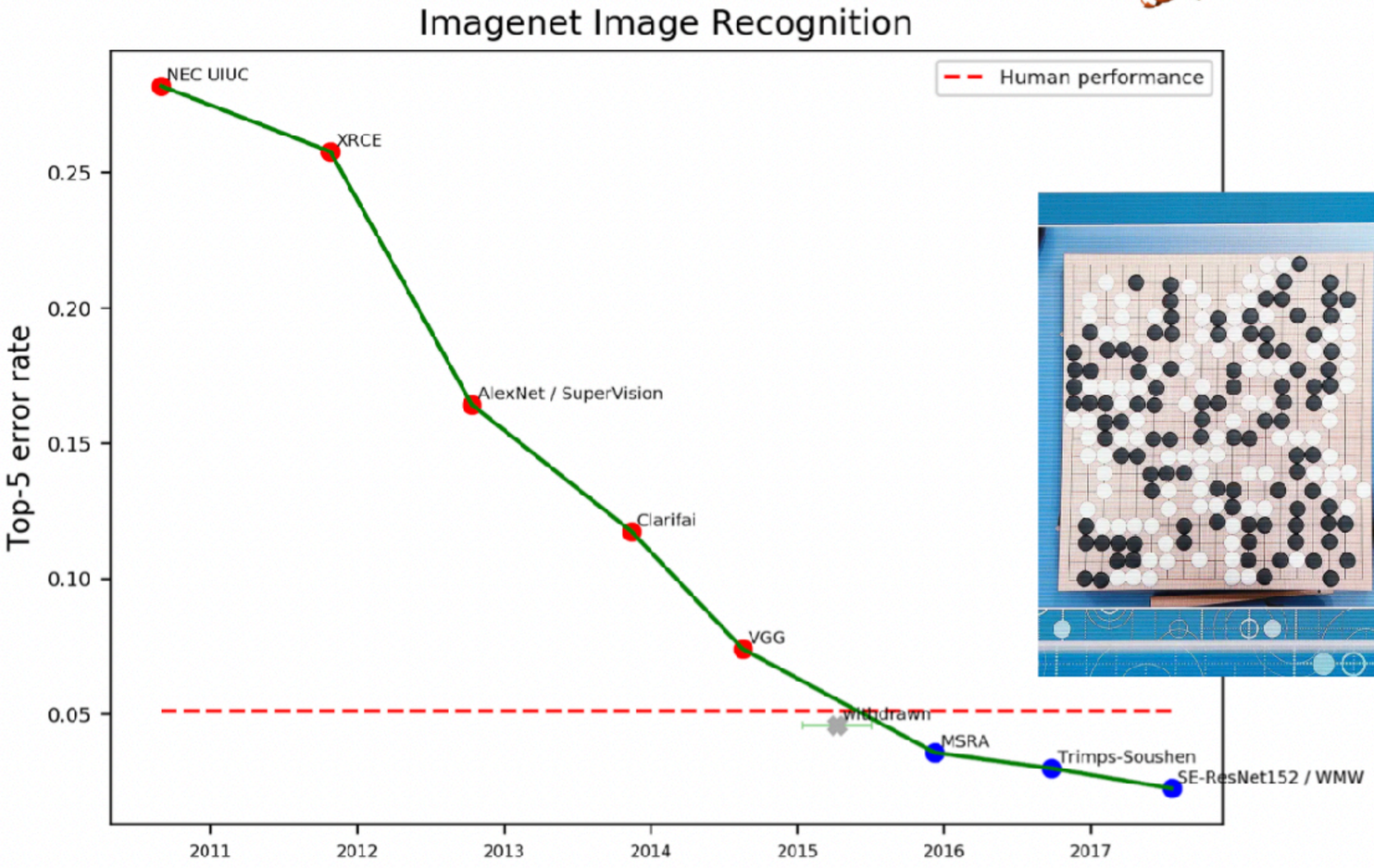
2012 - Крижевский и Хинтон публикуют первую успешную глубокую классификационную нейросеть AlexNet

- Вычислительных ресурсов!
- Хороших данных - собрали около 1.000.000 картинок
- Трюков и регуляризаций с третьей лекции:)
- Появления arxiv и github, например создатели AlexNet написали свою нейросеть на опенсурсной библиотеке CUDA



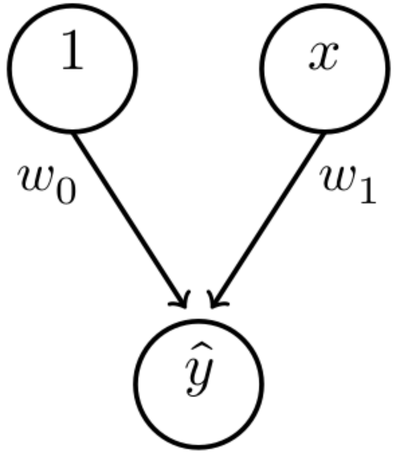
Deep learning summer

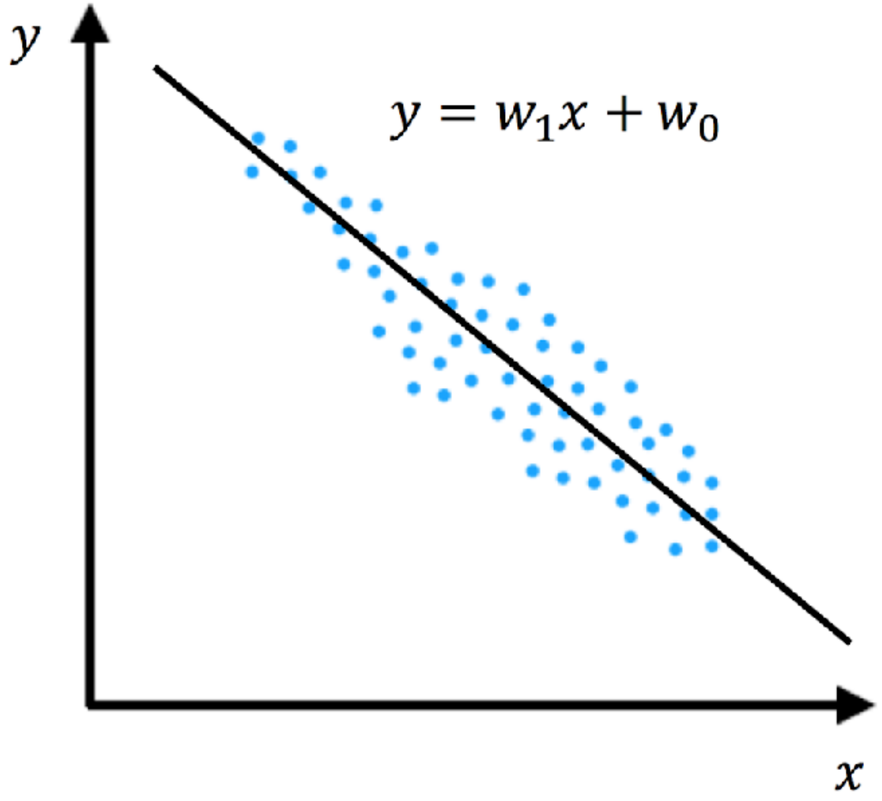




От регрессии к нейросетке

Линейная регрессия





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

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

$$y_i = (x_i, w)$$

Линейная регрессия (векторная форма)

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

Модель:

$$y = Xw$$

Оценка:

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

Прогноз:

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

Как обучить линейную регрессию?

- Нужно ввести штраф за ошибку:

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

- Не для всех функция потерь бывает аналитическое решение, например для MAE из-за модуля его нет:

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

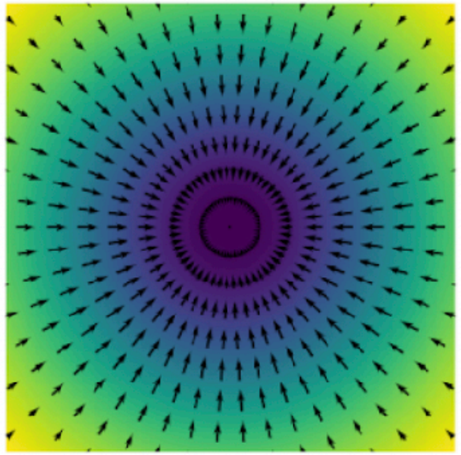
- Обычно модель обучают методом градиентного спуска

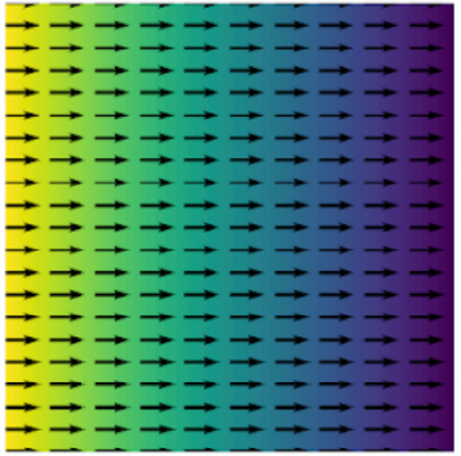
Градиентный спуск

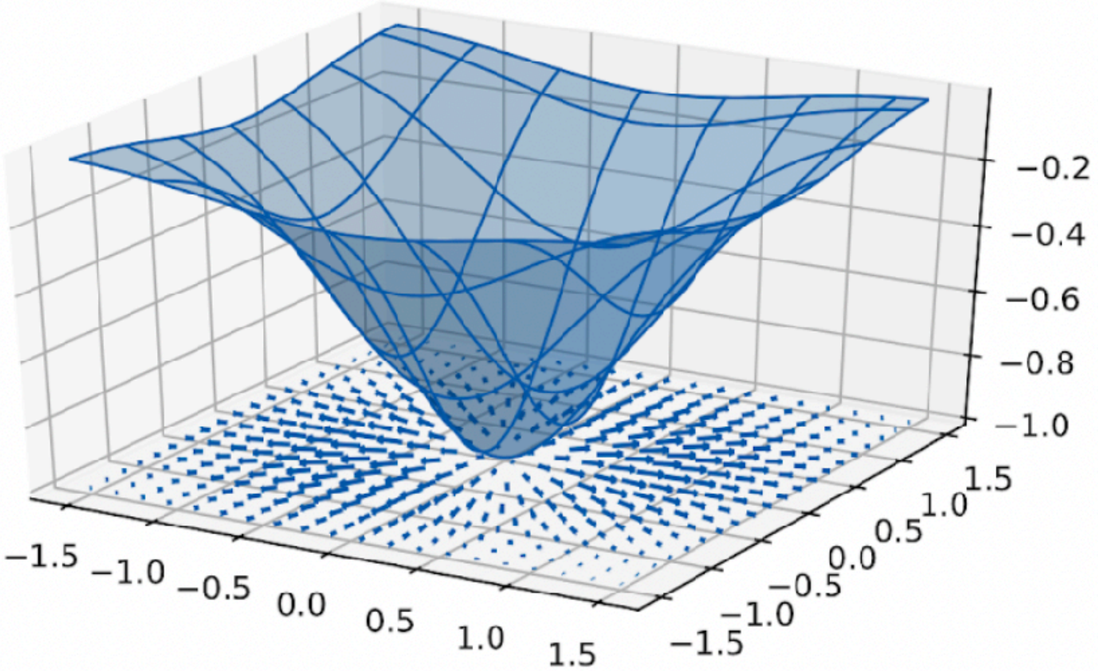
- Градиент:

$$\nabla_x f(x) \equiv \left(\frac{\partial f(x)}{\partial x_1}, \dots, \frac{\partial f(x)}{\partial x_d}\right)$$

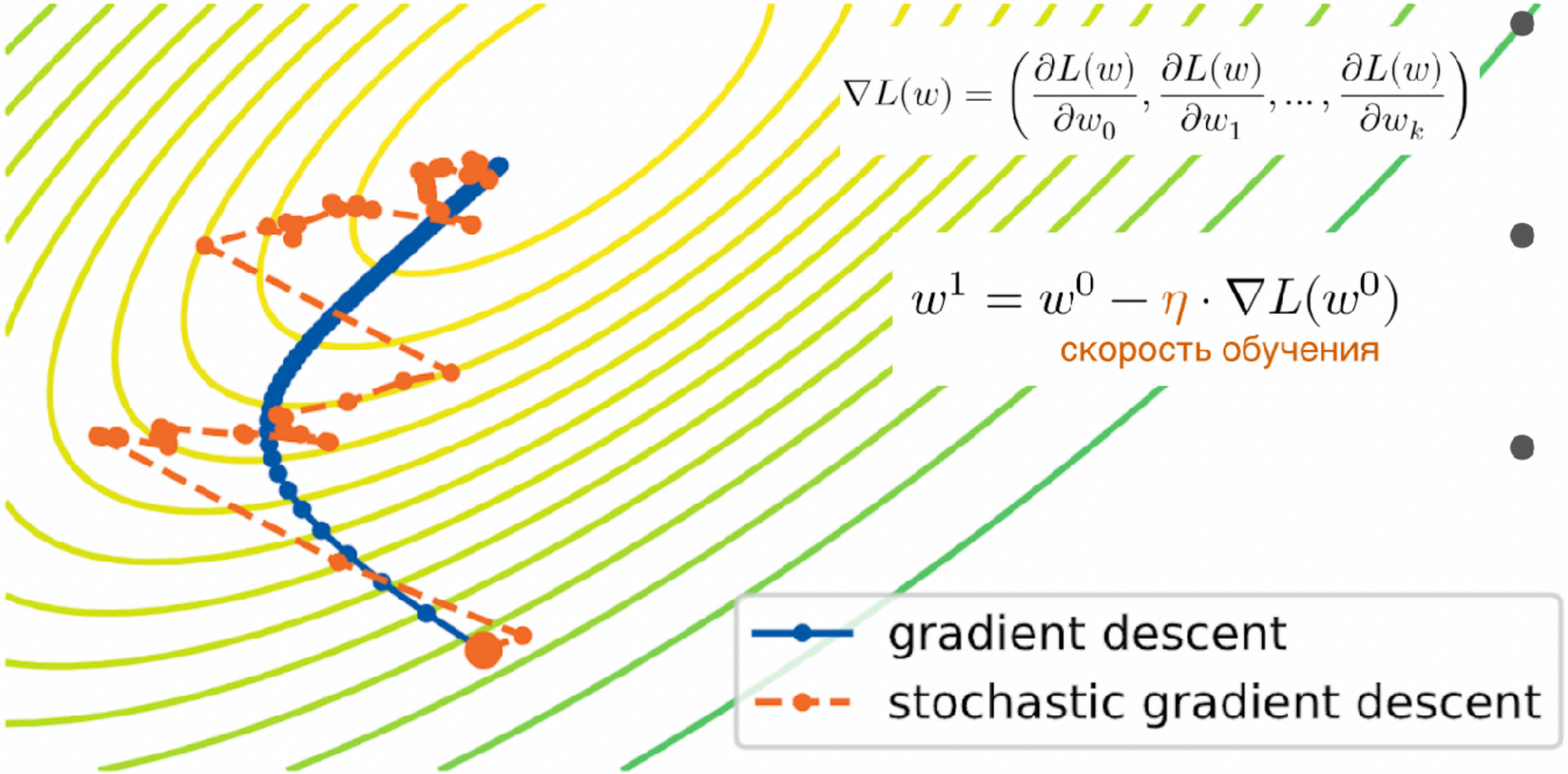
- Указывает направление наибольшего роста функции







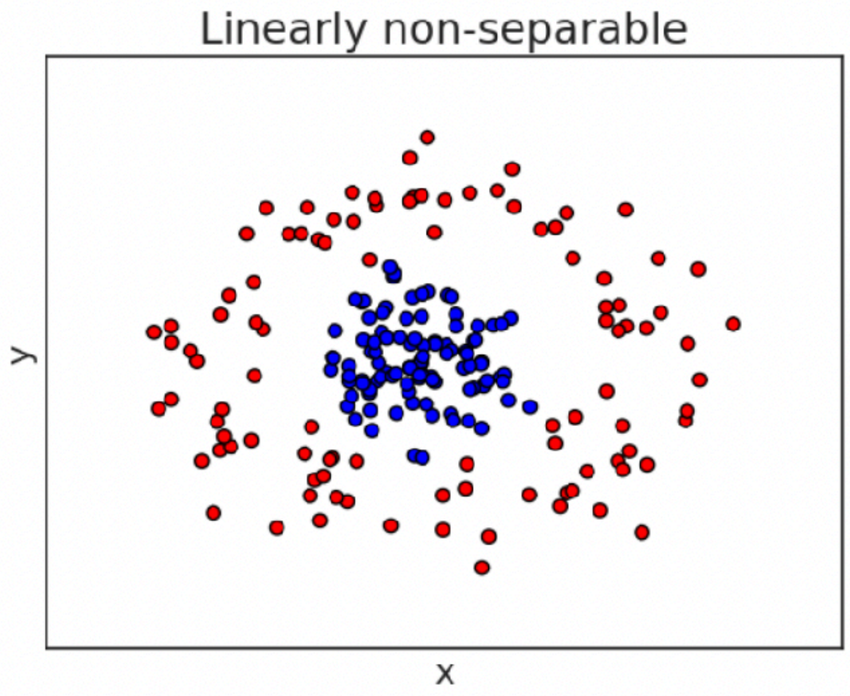
Градиентный спуск

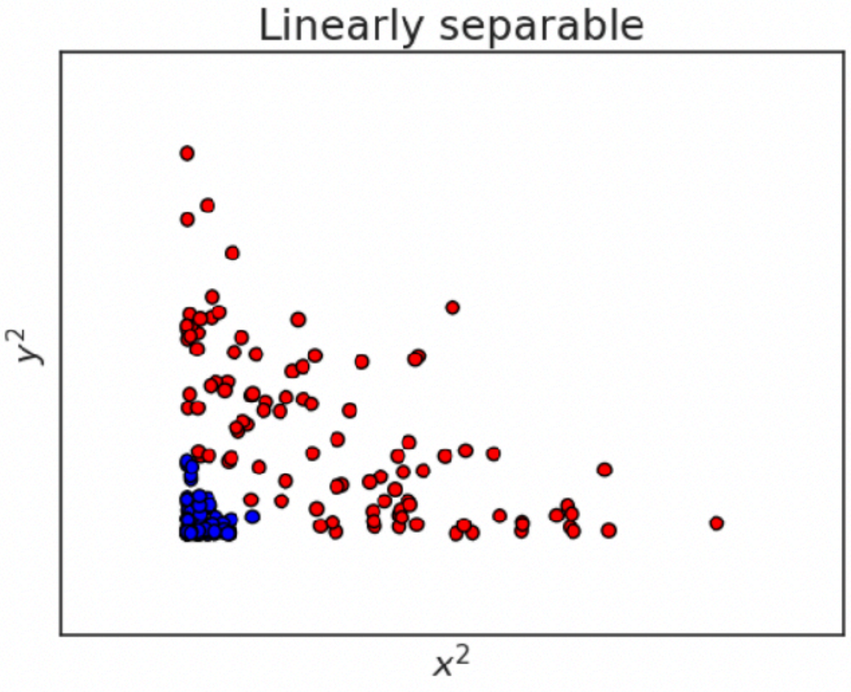


- Рассчитываем градиент функции потерь по весам
- Изменяем веса в соответствии с градиентом
- Можем делать это стохастично: по одному или нескольким сэмплам

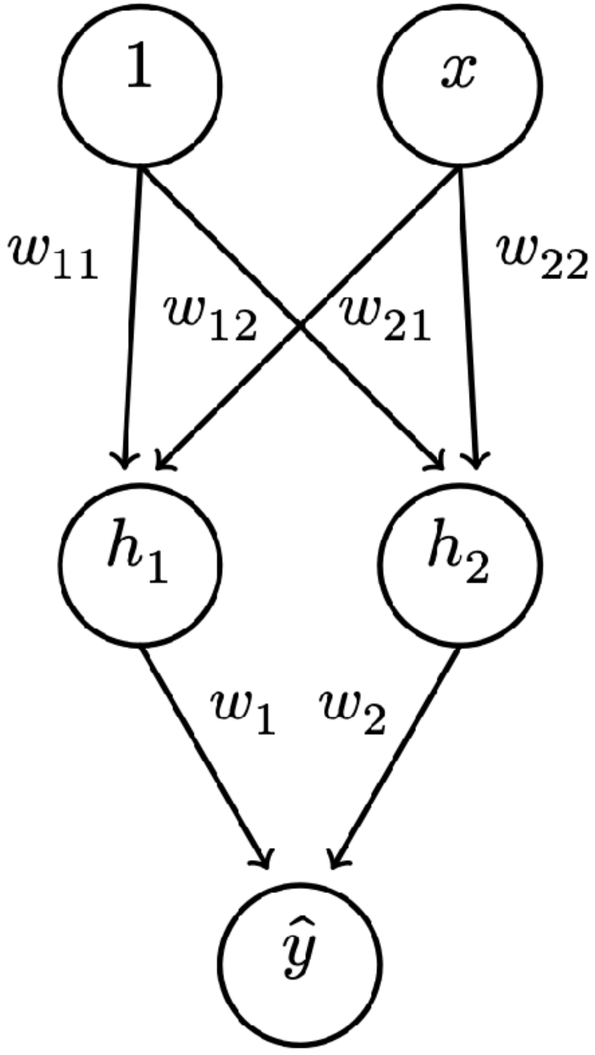
От линейной модели к MLP

- **Reminder:** линейные модели хорошо работают тогда когда пространство линейно разделимо
- Для того чтобы добиться этого часто необходимо было применять препроцессинг (например полиномиальные признаки)
- Можно ли сделать некий универсальный препроцессинг?





А что если...



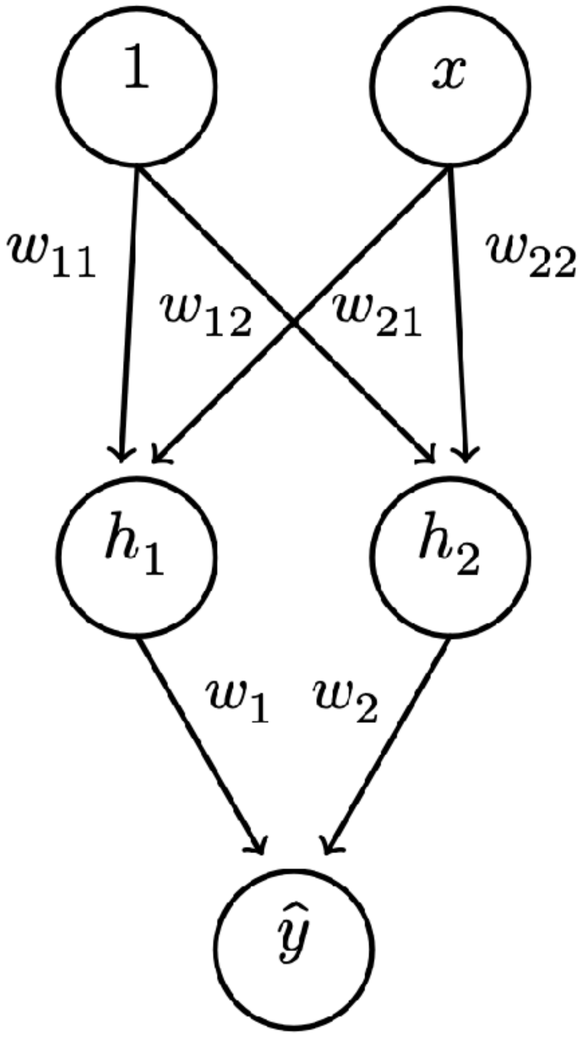
$$h_{1i} = w_{11} + w_{21} \cdot x_i$$
$$h_{2i} = w_{12} + w_{22} \cdot x_i$$

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

$$h = X \cdot W_1$$
$$y = h \cdot W_2$$

Норм идея?

А что если...

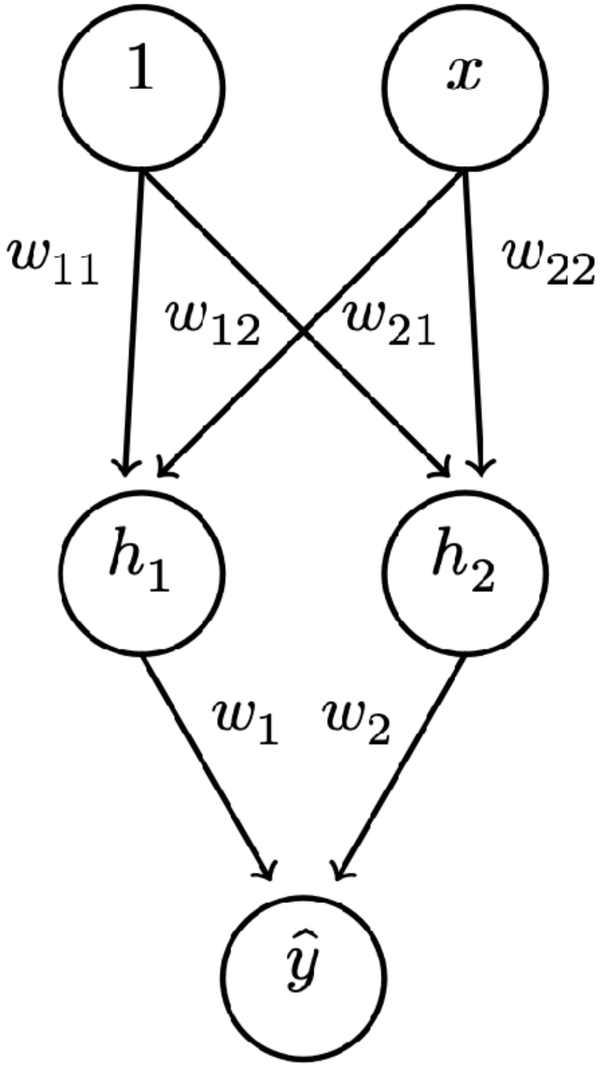


$$
\begin{aligned}
y &= w_1 \cdot h_1 + w_2 \cdot h_2 = \\
&= w_1 \cdot (w_{11} + w_{21} \cdot x) + w_2 \cdot (w_{12} + w_{22} \cdot x) = \\
&= \underbrace{(w_1 w_{11} + w_2 w_{12})}_{\gamma_1} + \underbrace{(w_1 w_{21} + w_2 w_{22})}_{\gamma_2} x
\end{aligned}
$$

Чёрт возьми! Опять линейность...

$$y = h \cdot W_2 = X \cdot W_1 \cdot W_2 = X \cdot A$$

А что если...



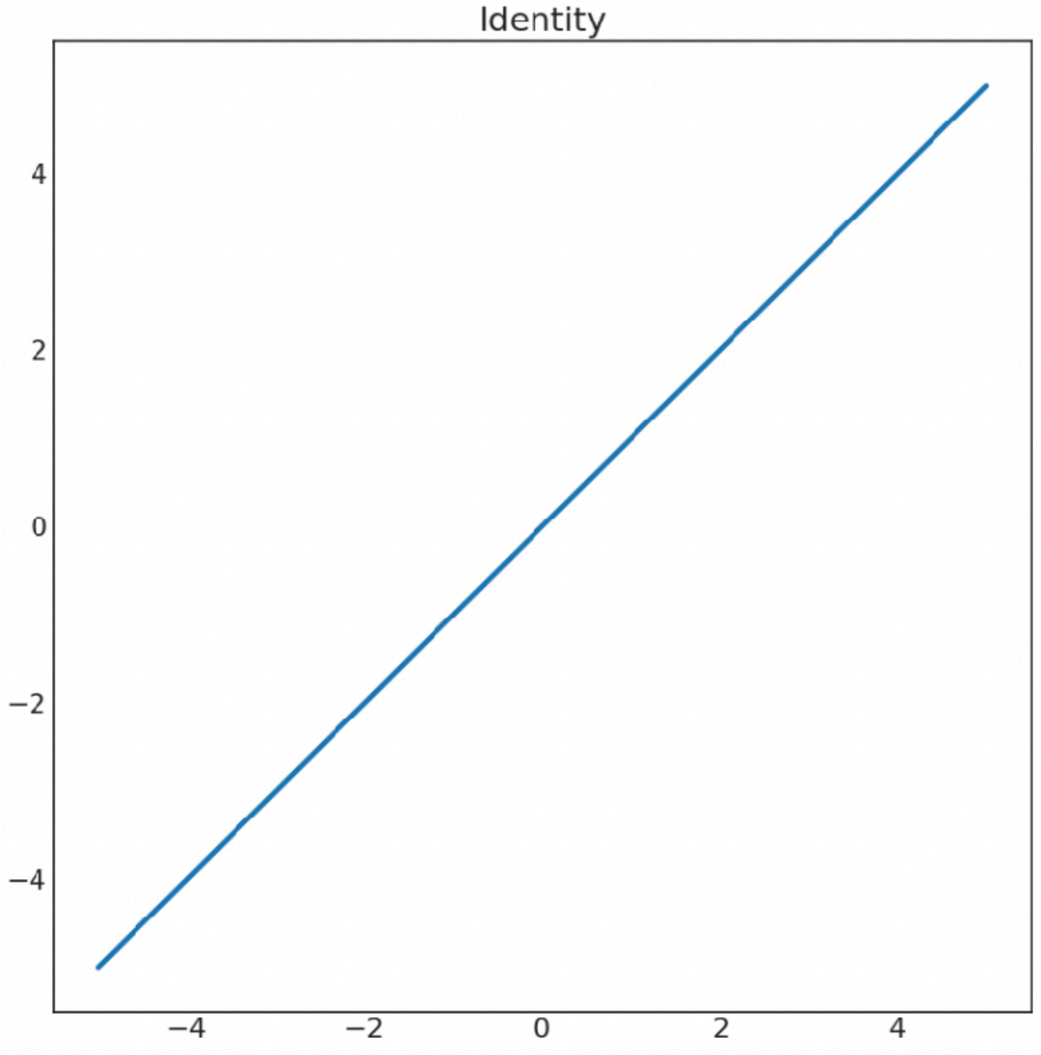
- Давайте добавим к скрытому состоянию какую-нибудь нелинейность:

$$h_{1i} = w_{11} + w_{21} \cdot x_i$$
$$h_{2i} = w_{12} + w_{22} \cdot x_i$$
$$y_i = w_1 \cdot f(h_{1i}) + w_2 \cdot f(h_{2i})$$

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

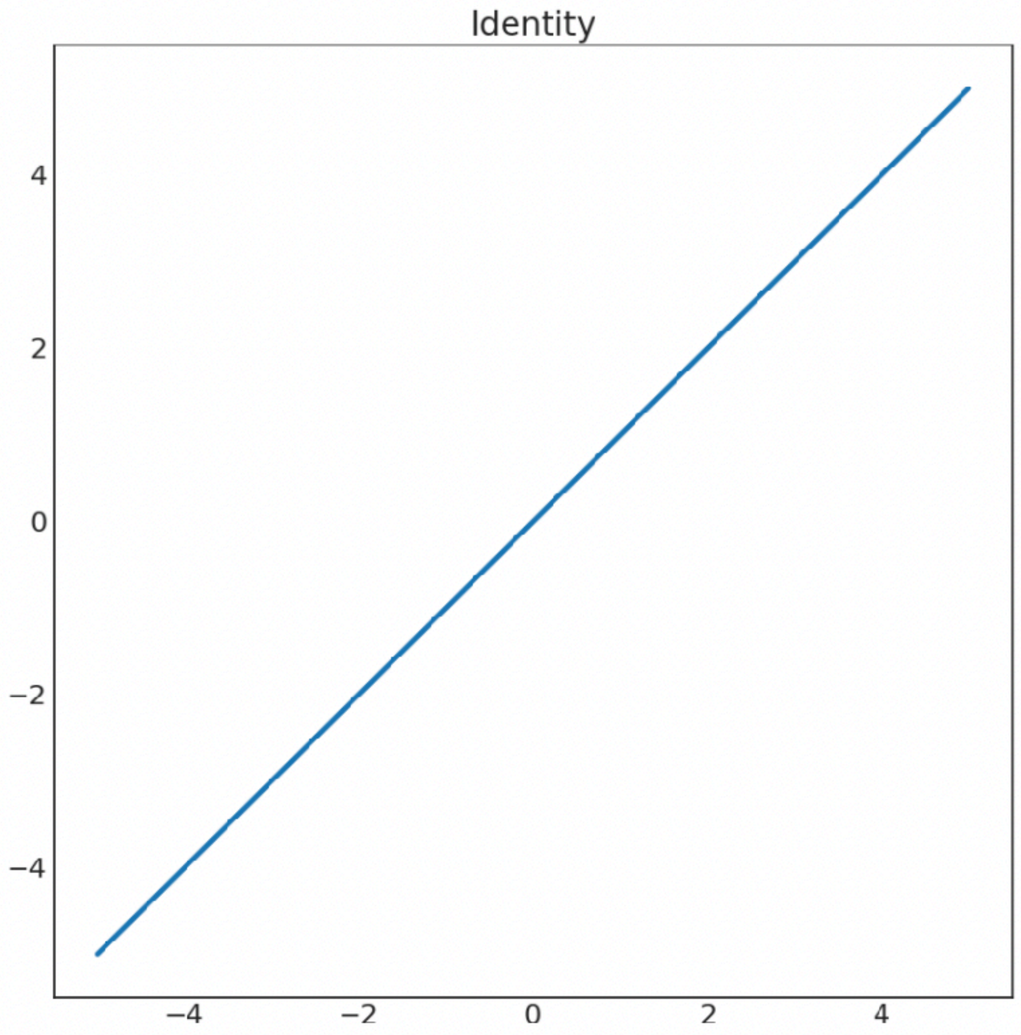
Функции активации

Как вы считаете, хорошая это активация или не очень?



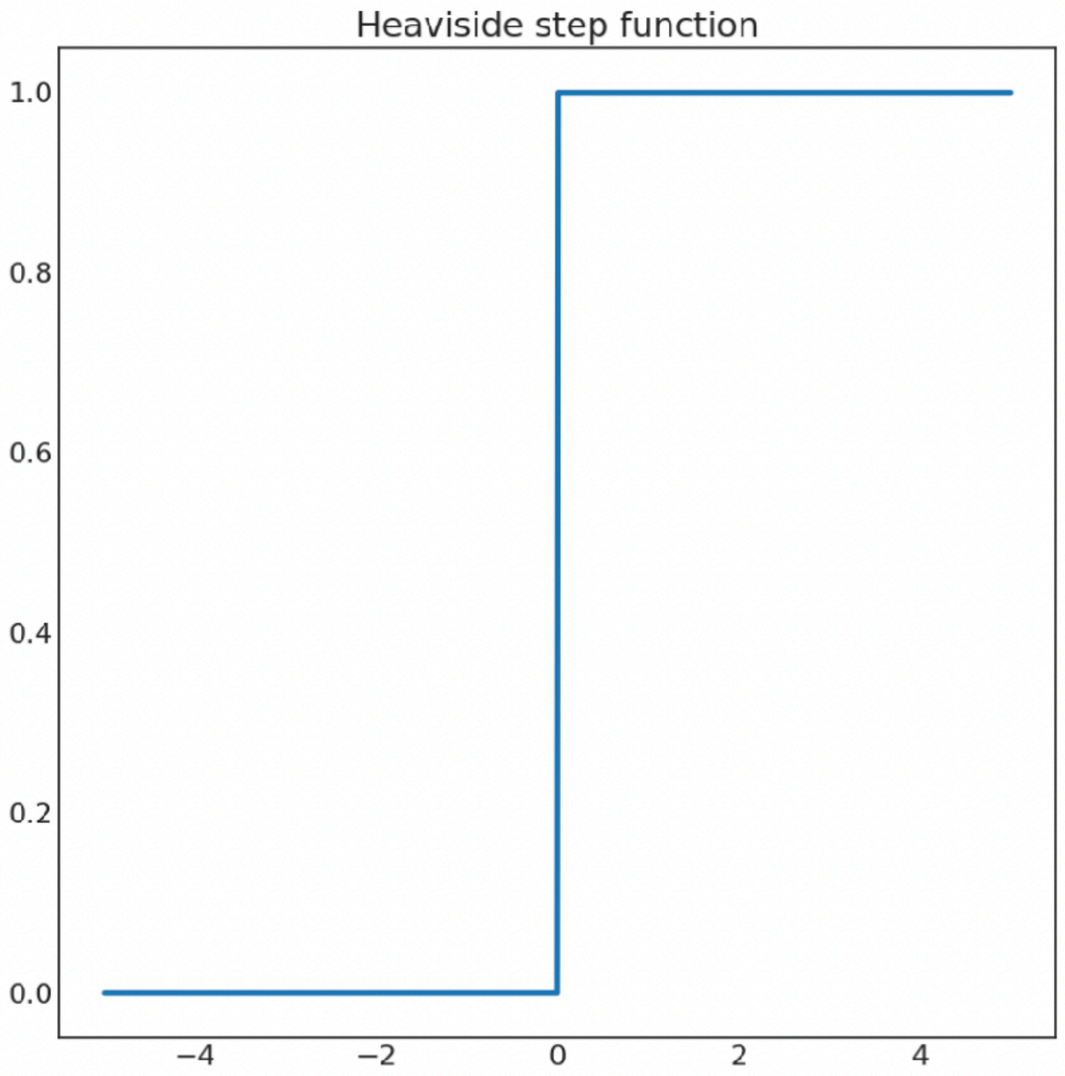
Функции активации

Правильно! Это вообще не нелинейность!



Функции активации: Heaviside step

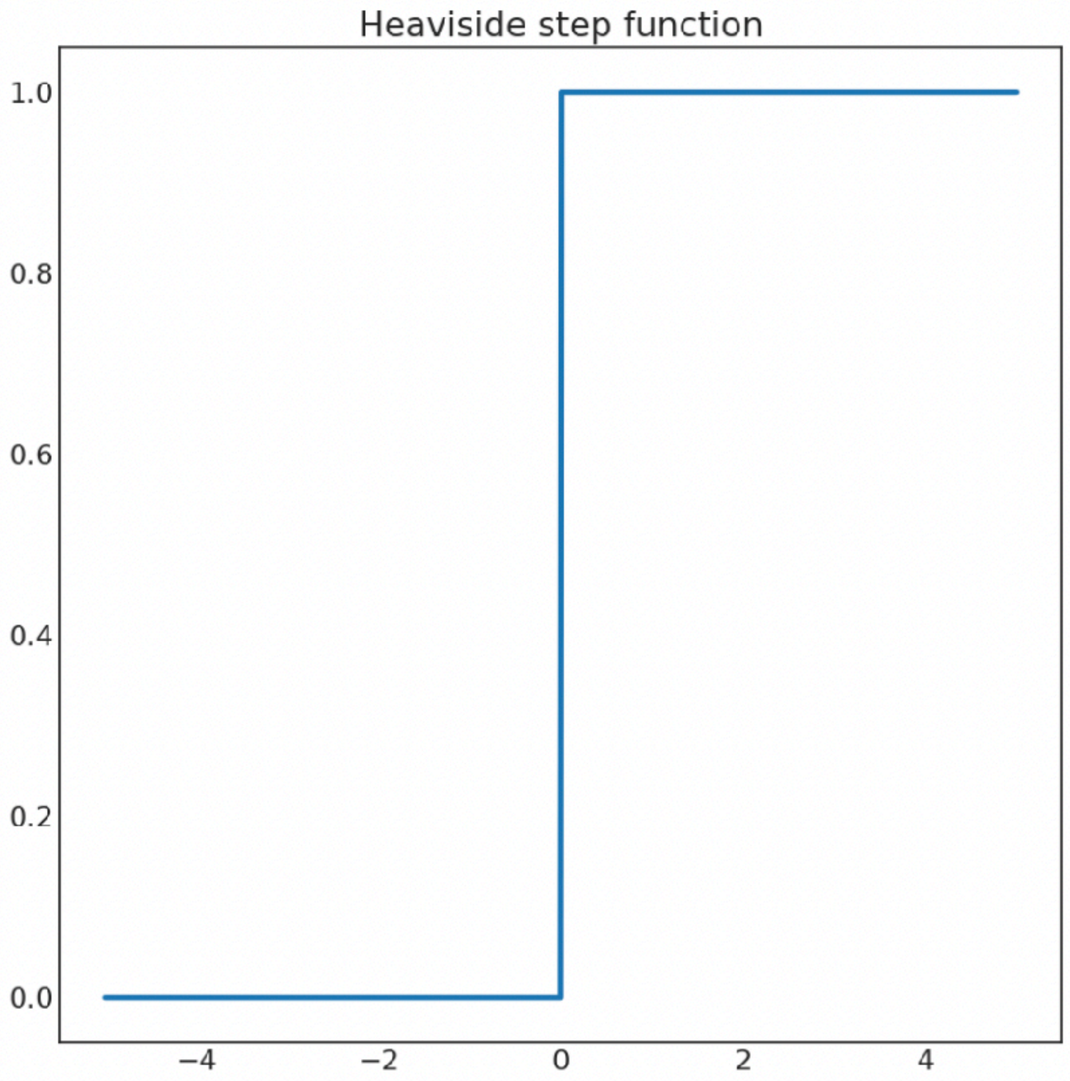
Как вы считаете, хорошая это активация или не очень?



Функции активации: Heaviside step

На самом деле неплохая, но только если у вас один скрытый слой… В чем проблема?

- Градиент равен нулю везде кроме одной точки*
- Зато очень просто вычислять



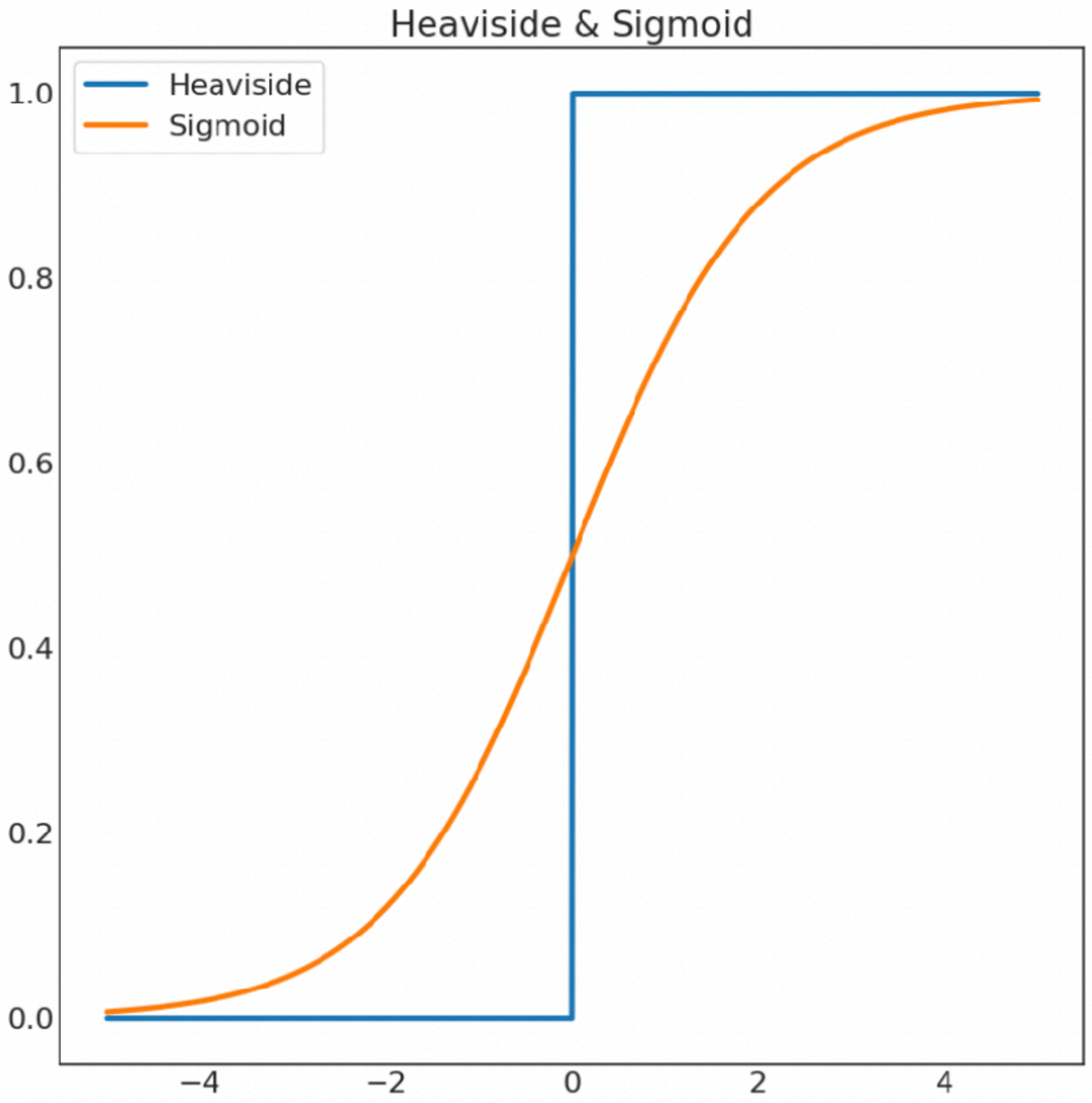
* а мы всё ещё хотим использовать градиентный спуск

Функции активации: сигмоида

Построим сглаженную версию “ступеньки”.

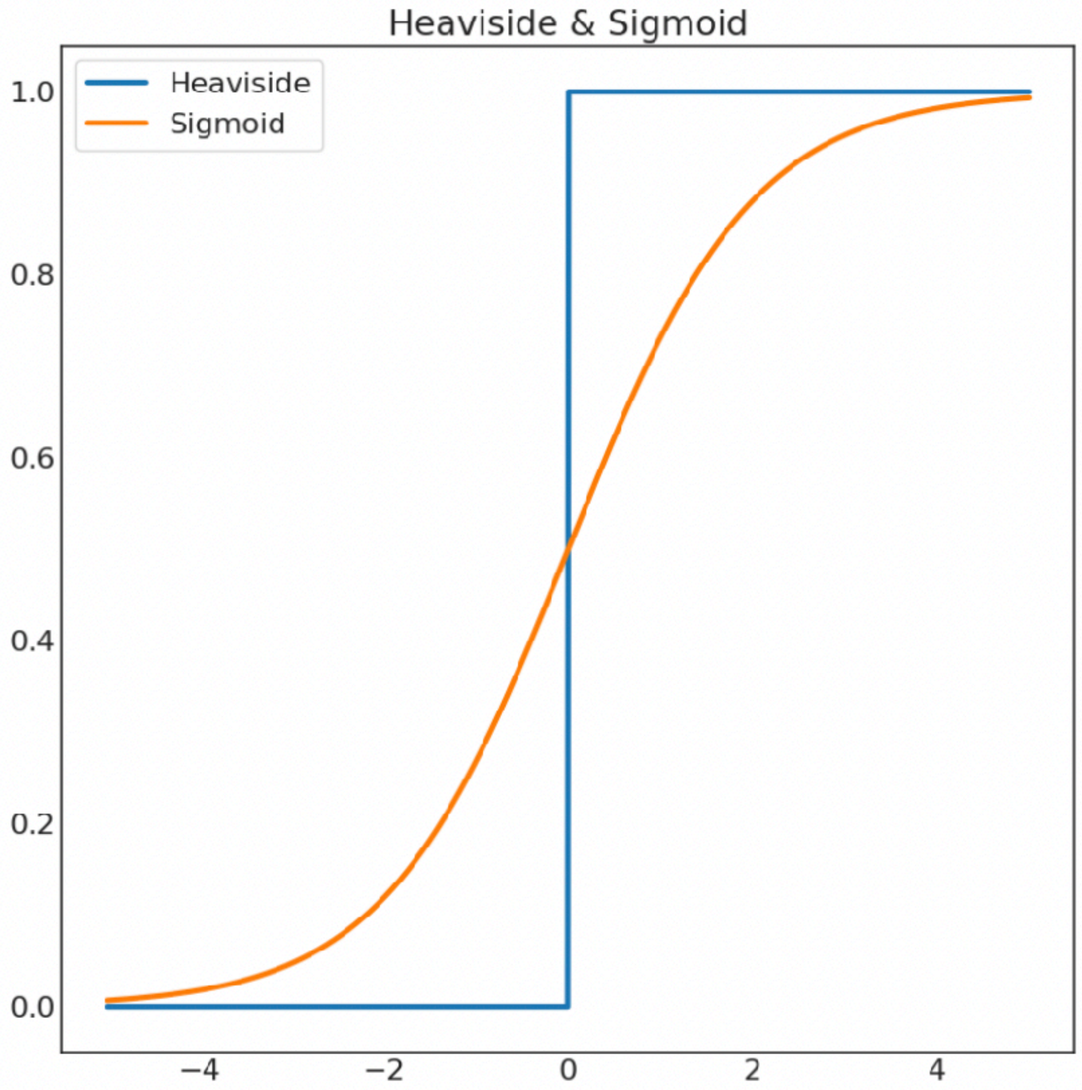
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Как вы считаете, хорошая это активация или не очень?



Функции активации: сигмоида

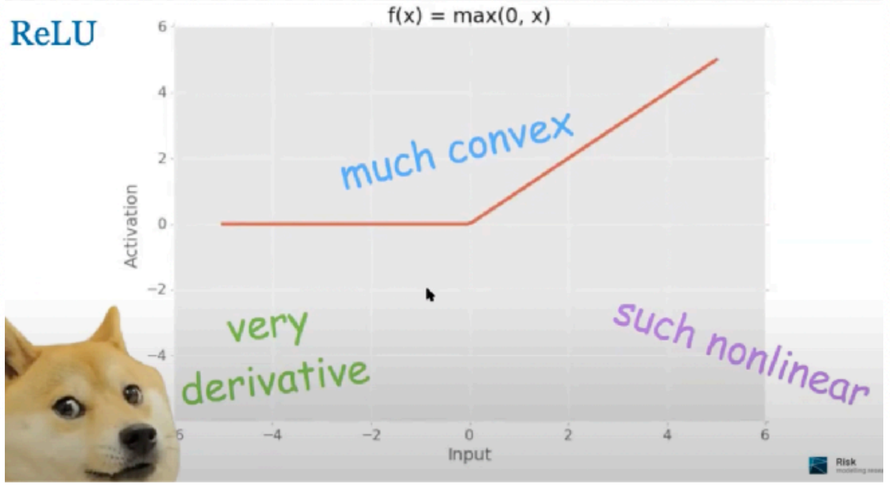
- Градиент есть везде и он не ноль
- Однако главная проблема - затухание с увеличением количества слоев
- Поэтому больших сеток не собрать, а как теперь известно - это то что лучше всего работает
- Смотрим локально только на изменение отклика около нуля

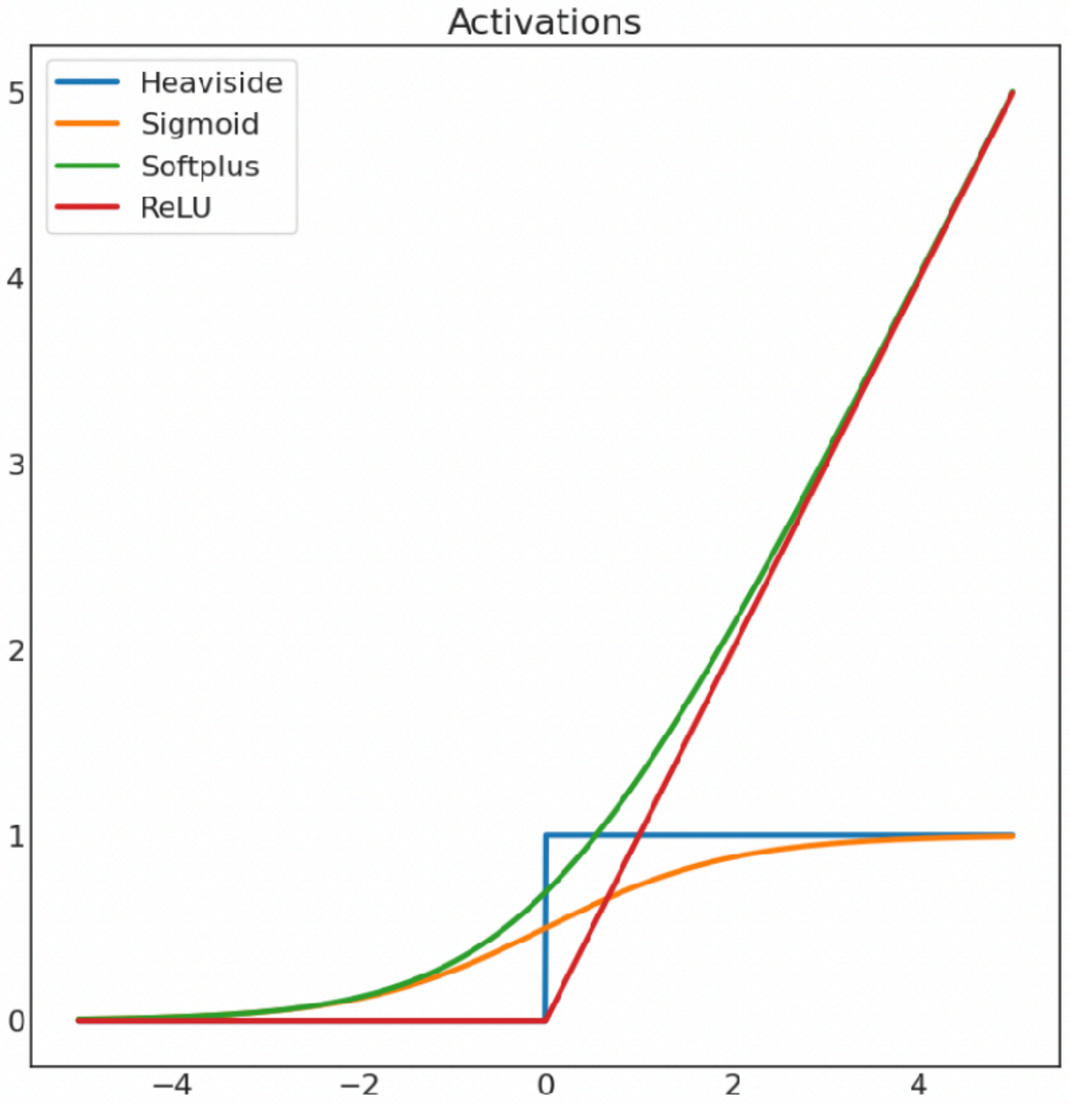


Функции активации: ReLU

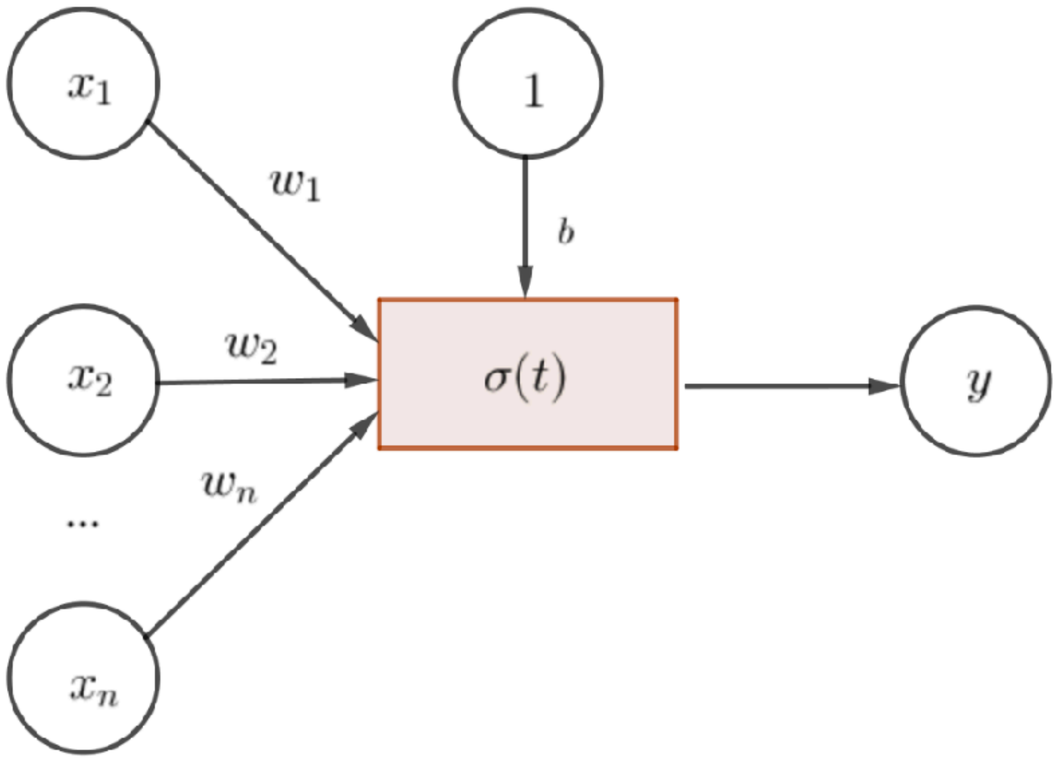
- Рассмотрим: $\mathrm{ReLU}(x) = \max(0, x)$

- Однако её гораздо проще вычислять и что более важно, гораздо проще рассчитывать градиенты



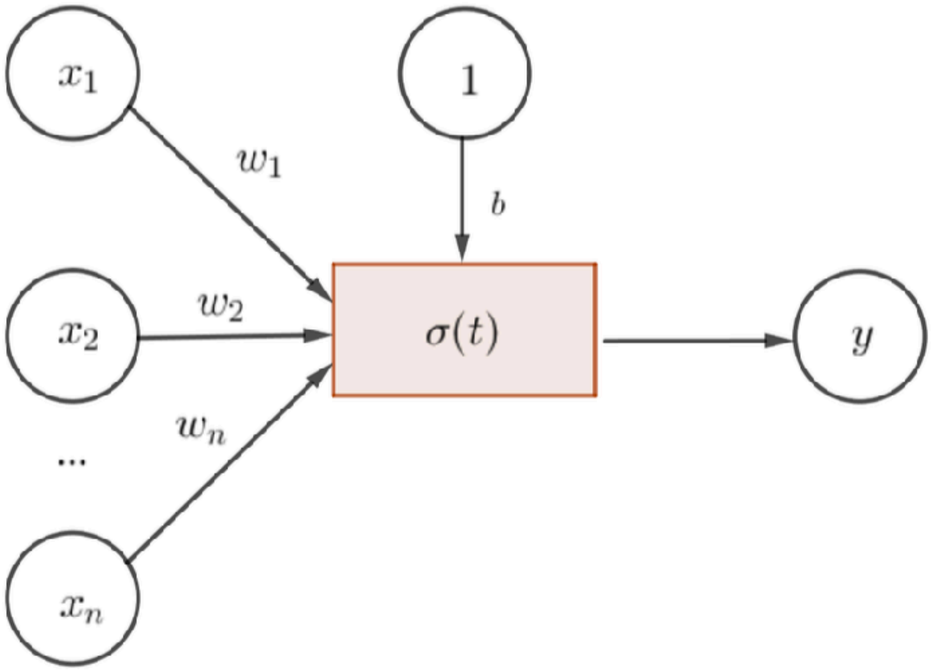


Функция активации



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

Линейная регрессия

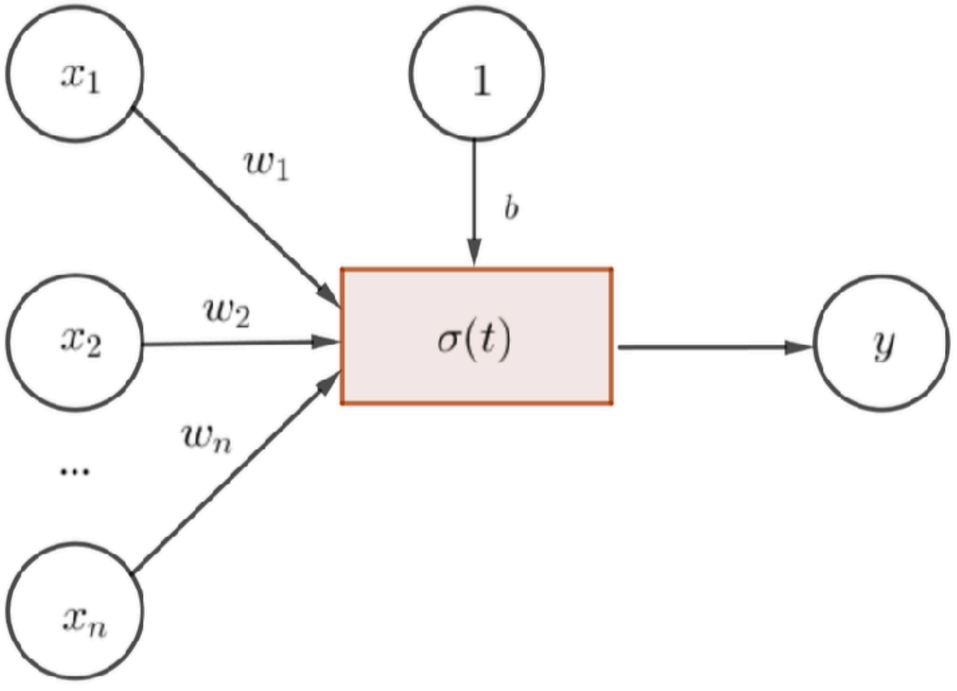


Нейрон с линейной функции активации — это линейная регрессия...

$$\sigma(t) = t$$

$$y = w_0 + w_1 \cdot x_1 + ... + w_n \cdot x_n$$

Логистическая регрессия

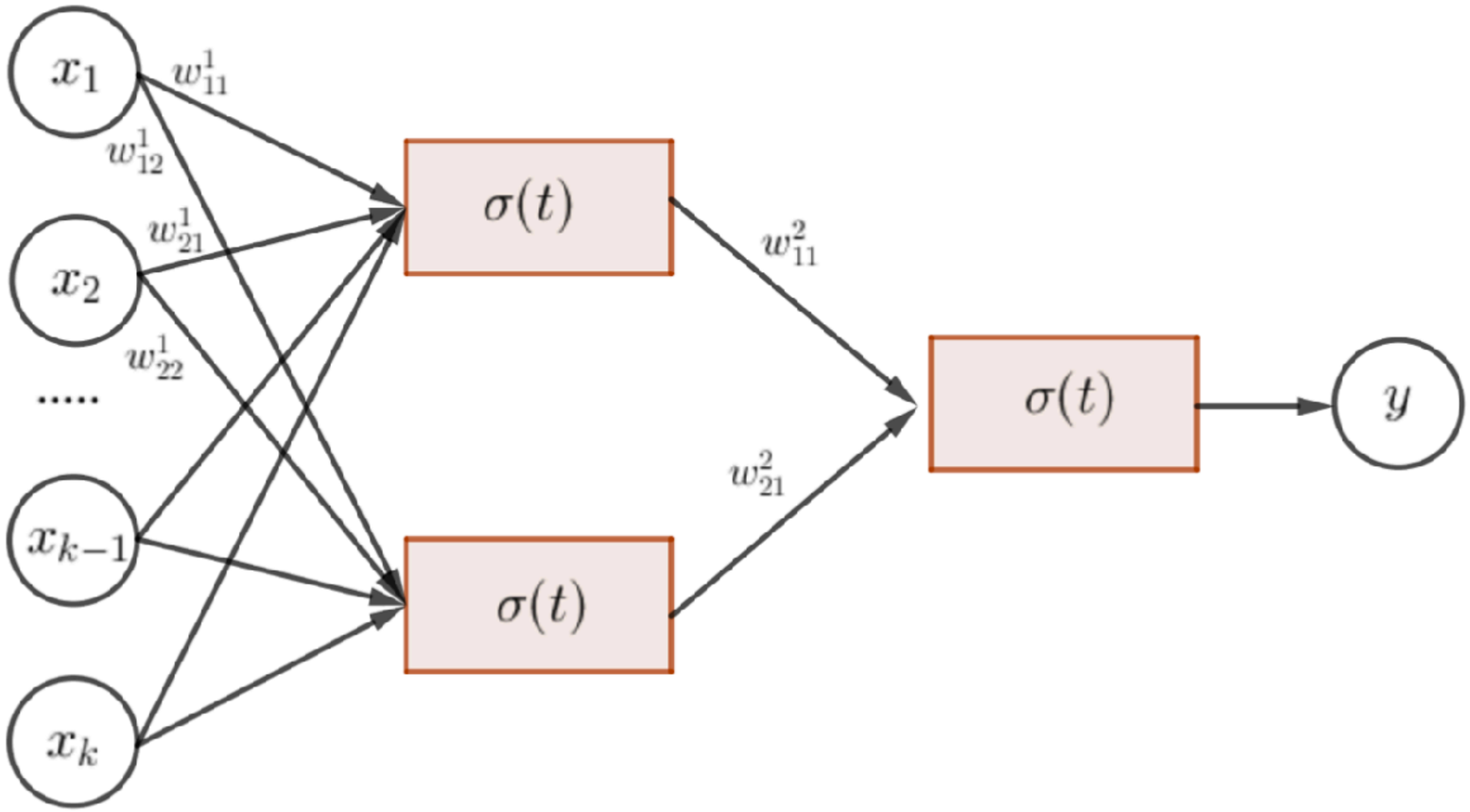


Нейрон с сигмоидом в качестве функции активации — это логистическая регрессия...

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

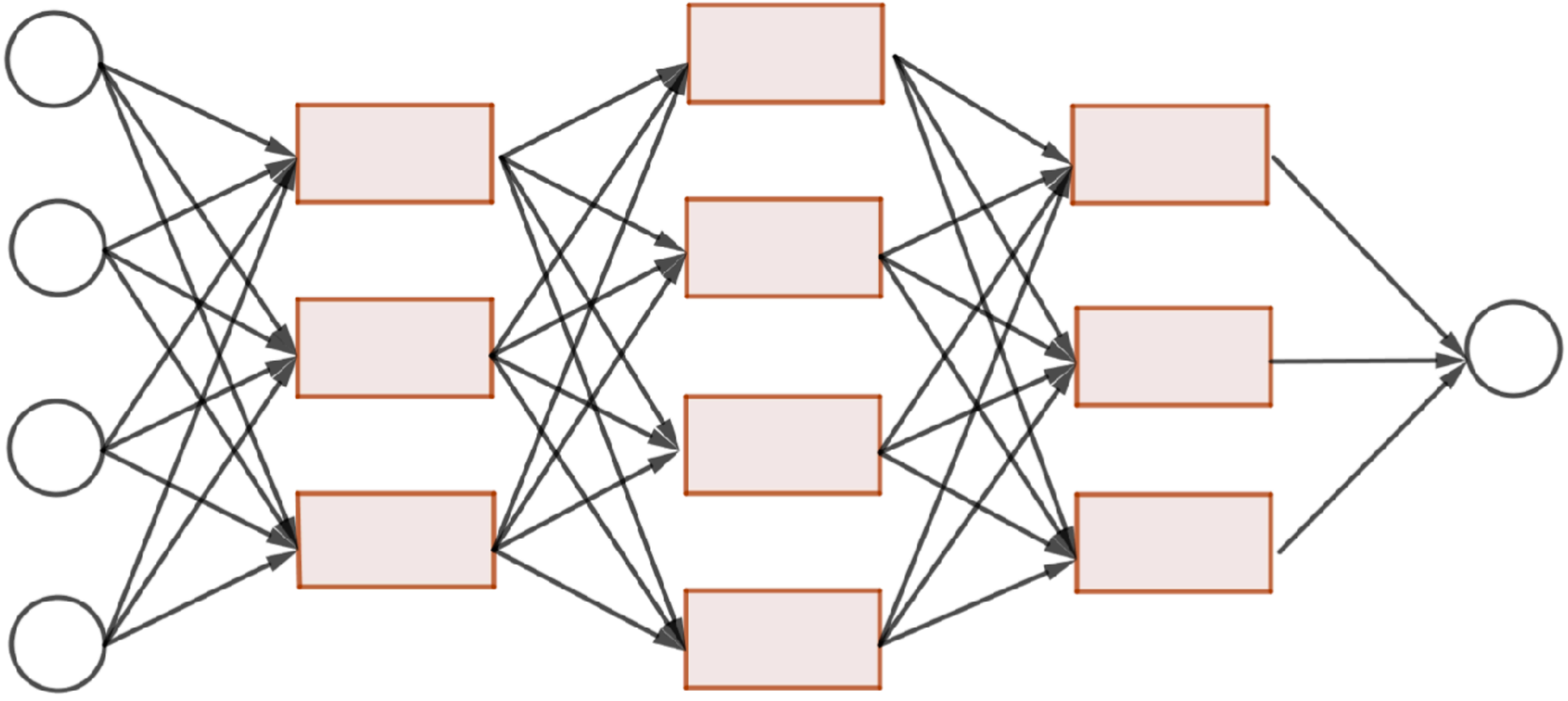
$$P(y = 1 \mid x) = \sigma(w_0 + w_1 \cdot x_1 + \ldots + w_n \cdot x_n)$$

MLP (multi-layer perceptron)



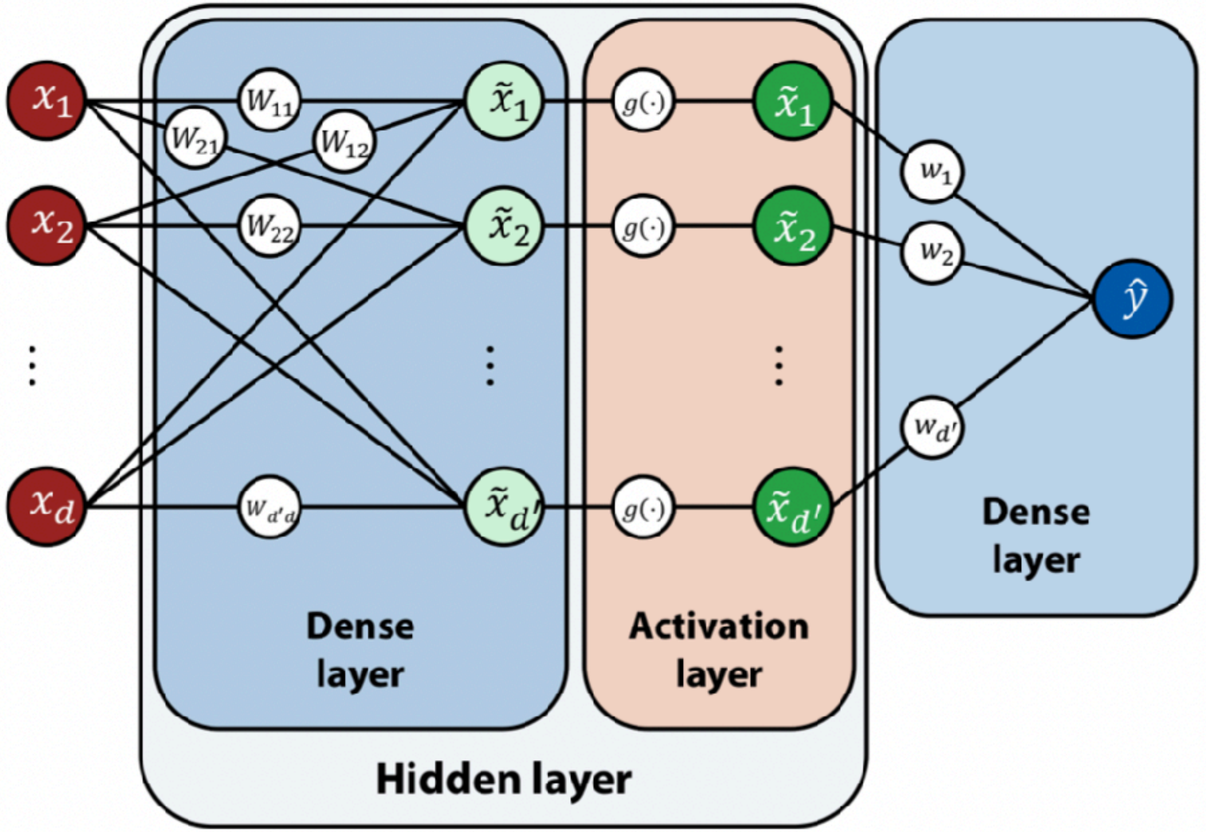
$$h = \sigma(X \cdot W)$$
$$y = \sigma(h \cdot W)$$

Армия из регрессий



Универсальный аппроксиматор

- 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
- Для этого может потребоваться **“широкое” скрытое представление**



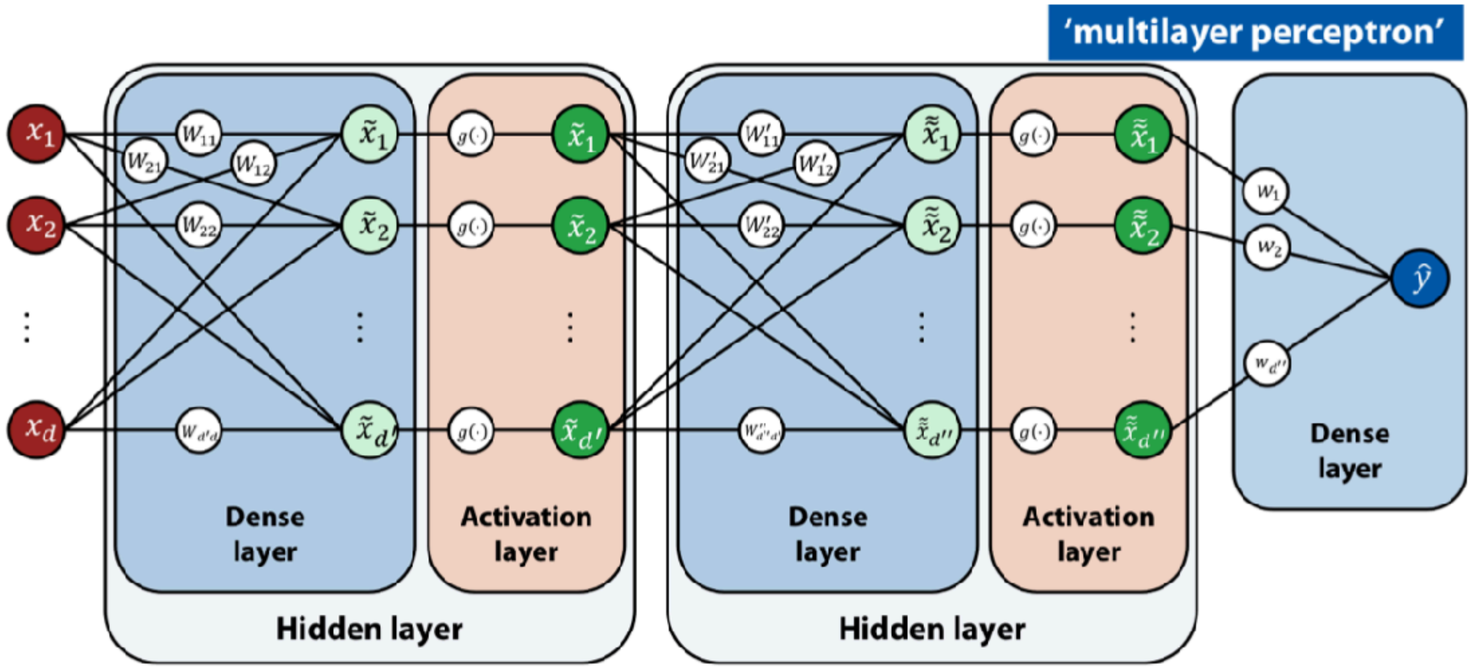
Графическое доказательство теоремы:
http://neuralnetworksanddeeplearning.com/chap4.html

Многослойные нейронные сети

Поэтому на практике используют **более “глубокие” сети**:

- С меньшими вычислительными затратами достигают такой же выразительности

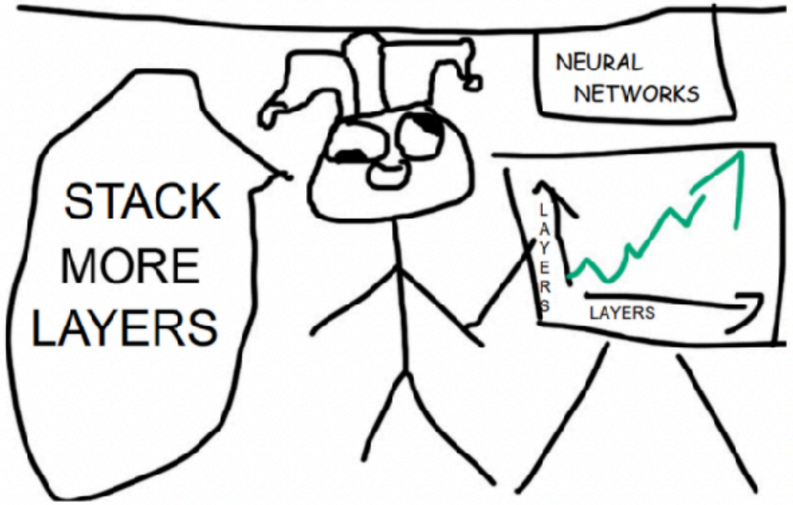
- Строятся скрытые признаки разного уровня абстрактности

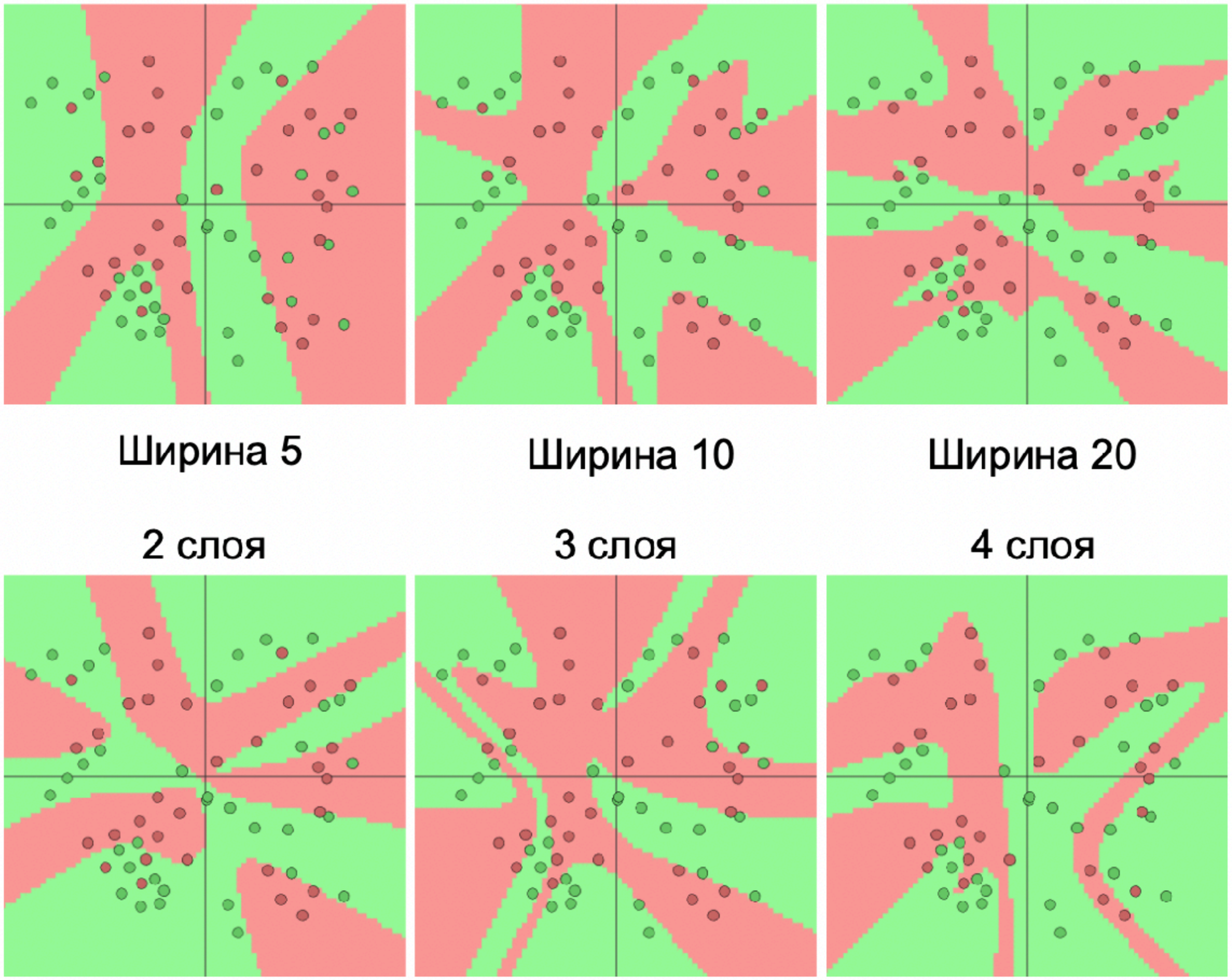


Наглядный пример:

Обучим координатную классификацию: с одним скрытым слоем разной ширины и со скрытым слоем ширины 5, разной глубины.

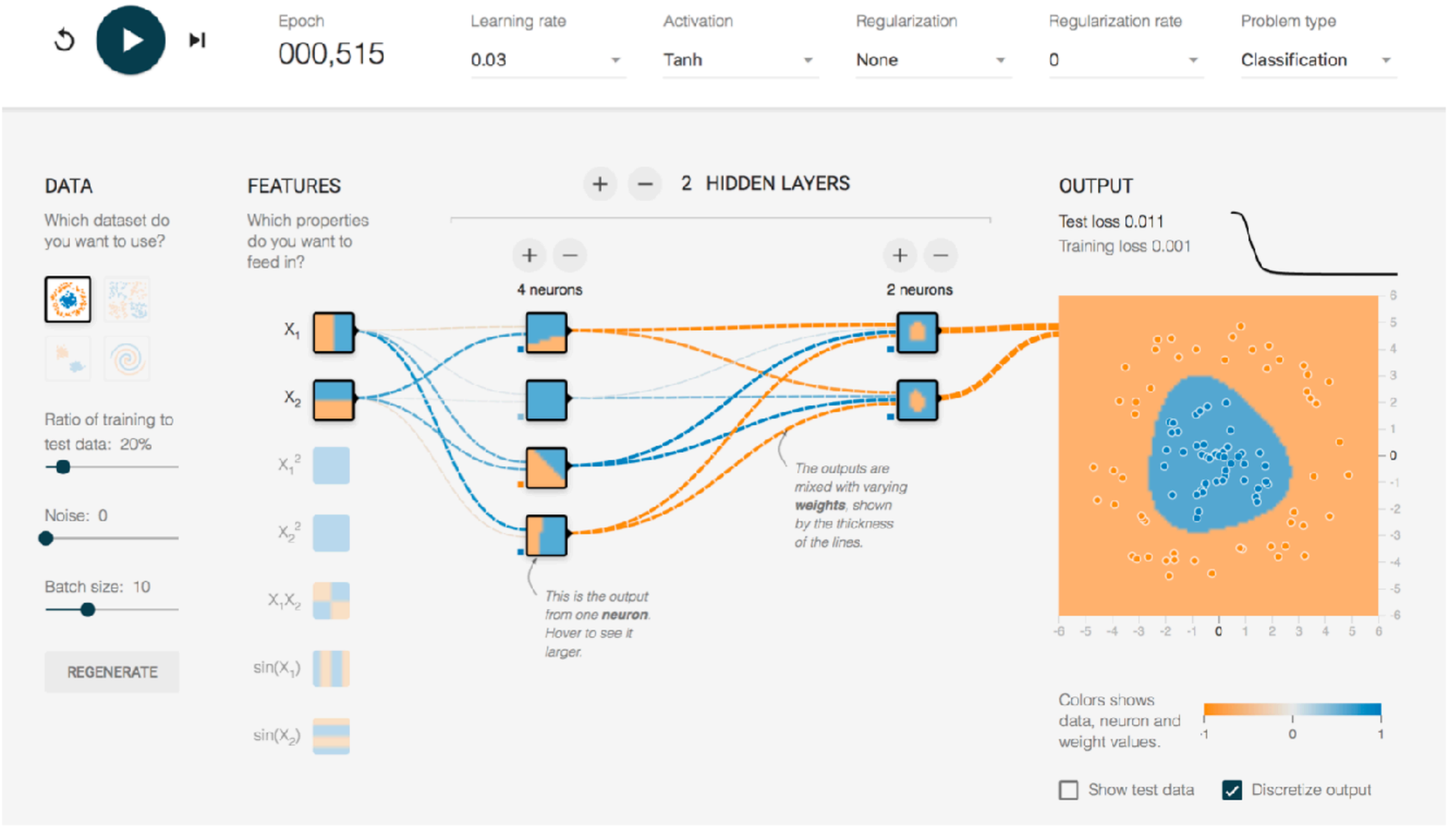
Вывод:





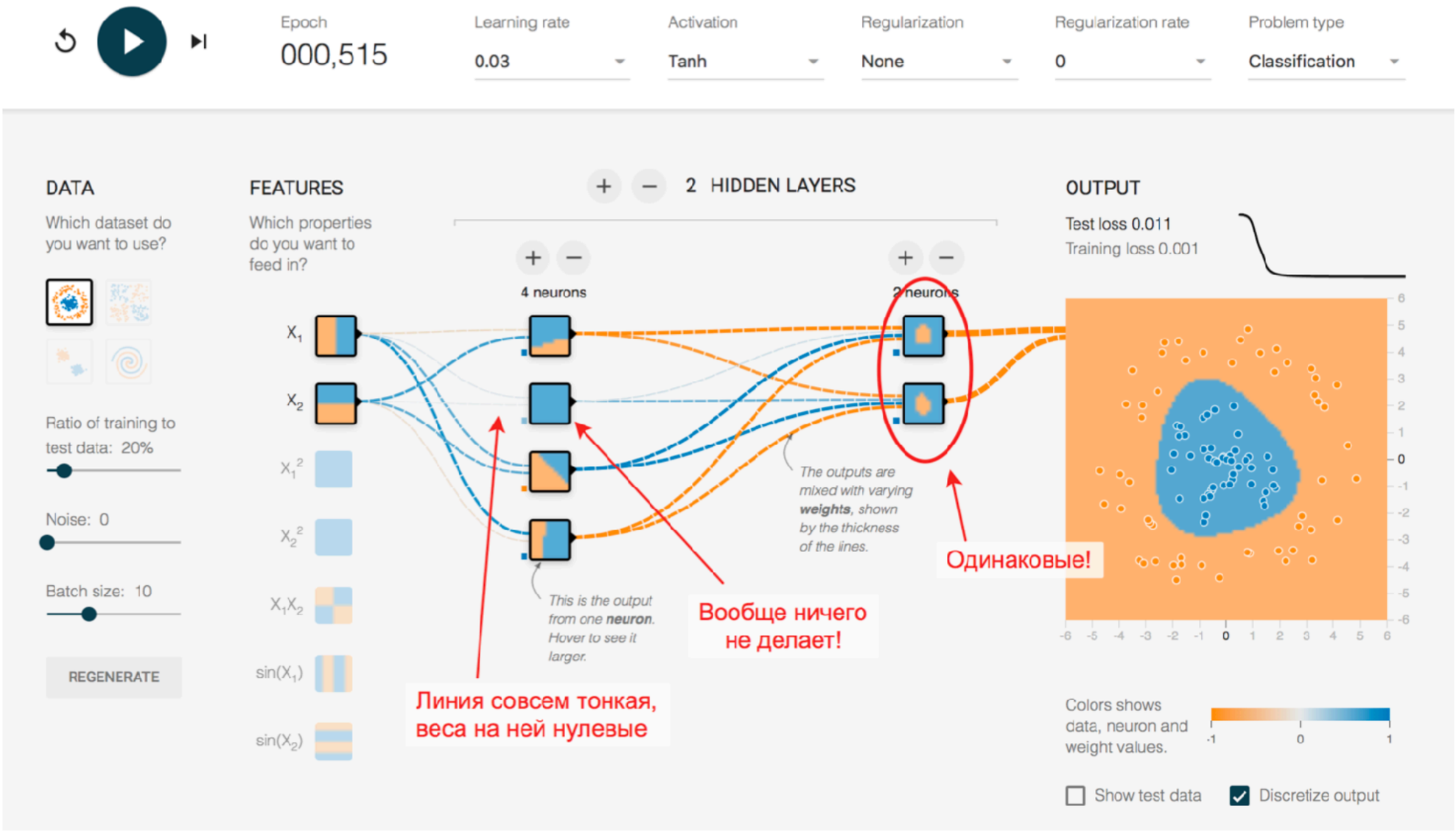
Demo from: https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

MLP не отходя от браузера



http://playground.tensorflow.org

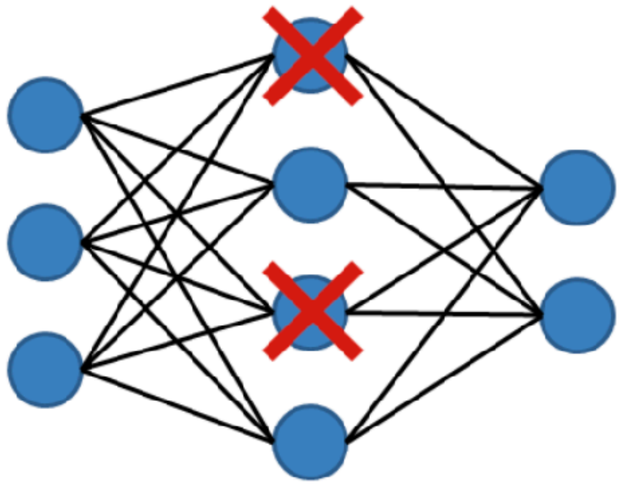
MLP не отходя от браузера



Слои бывают разными

- Слой, который просто взвешивает входы называется полносвязным.

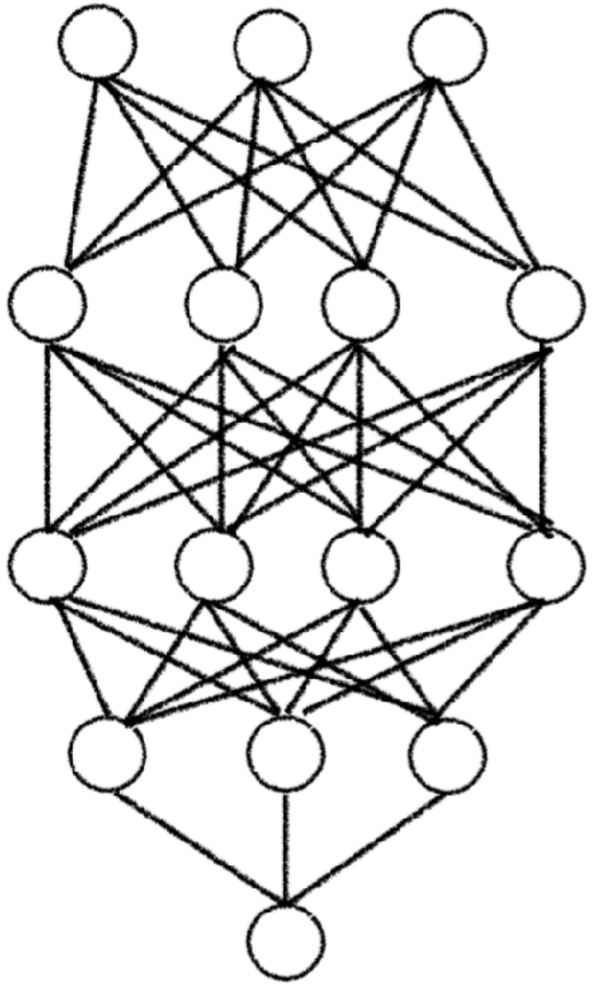
- Слои бывают очень разными. Например, Dropout: с вероятностью p отключаем нейрон. Такой слой препятствует переобучению и делает нейроны более устойчивыми к случайным возмущениям.



Функции активации бывают разными

Название функции	Формула $f(x)$	Производная $f'(x)$
Логистический сигмоид σ	$\frac{1}{1+e^{-x}}$	$f(x)\,(1-f(x))$
Гиперболический тангенс tanh	$\frac{e^x-e^{-x}}{e^x+e^{-x}}$	$1-f^2(x)$
SoftSign	$\frac{x}{1+\lvert x\rvert}$	$\frac{1}{(1+\lvert x\rvert)^2}$
Ступенька (функция Хевисайда)	$\begin{cases} 0, & x<0 \\ 1, & x\geq 0 \end{cases}$	0
SoftPlus	$\log(1+e^x)$	$\frac{1}{1+e^{-x}}$
ReLU	$\begin{cases} 0, & x<0 \\ x, & x\geq 0 \end{cases}$	$\begin{cases} 0, & x<0 \\ 1, & x\geq 0 \end{cases}$
Leaky ReLU, Parameterized ReLU	$\begin{cases} ax, & x<0 \\ x, & x\geq 0 \end{cases}$	$\begin{cases} a, & x<0 \\ 1, & x\geq 0 \end{cases}$
ELU	$\begin{cases} \alpha\,(e^x-1)\,, & x<0 \\ x, & x\geq 0 \end{cases}$	$\begin{cases} f(x)+\alpha, & x<0 \\ 1, & x\geq 0 \end{cases}$

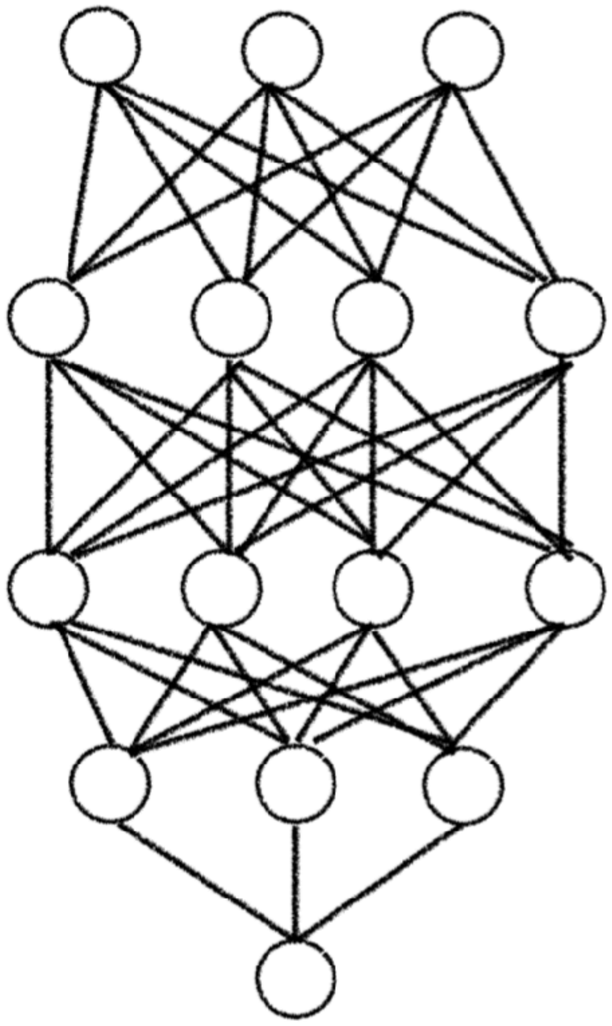
Архитектуры бывают разными



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	

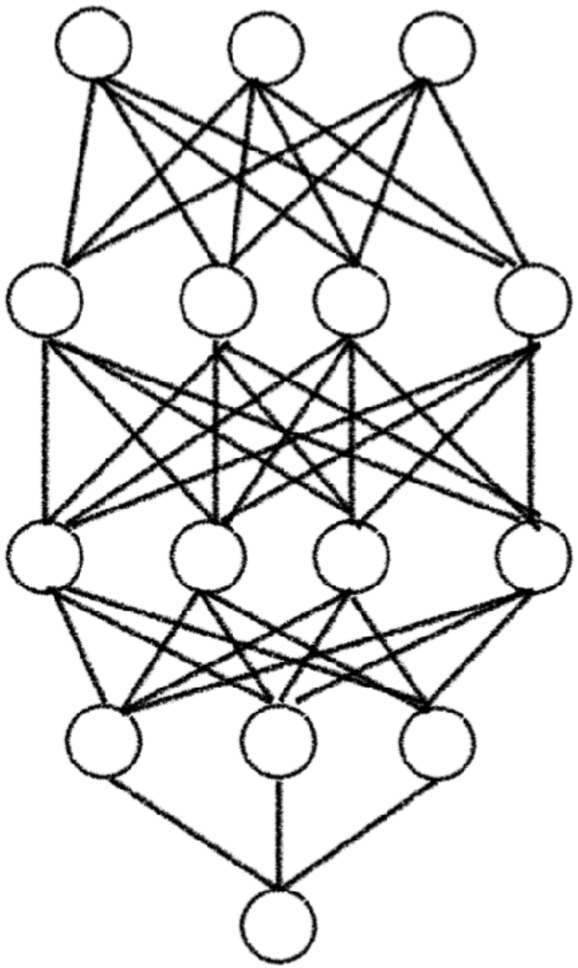
Каждый слой — просто функция, каждая сетка — конструктор LEGO

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	

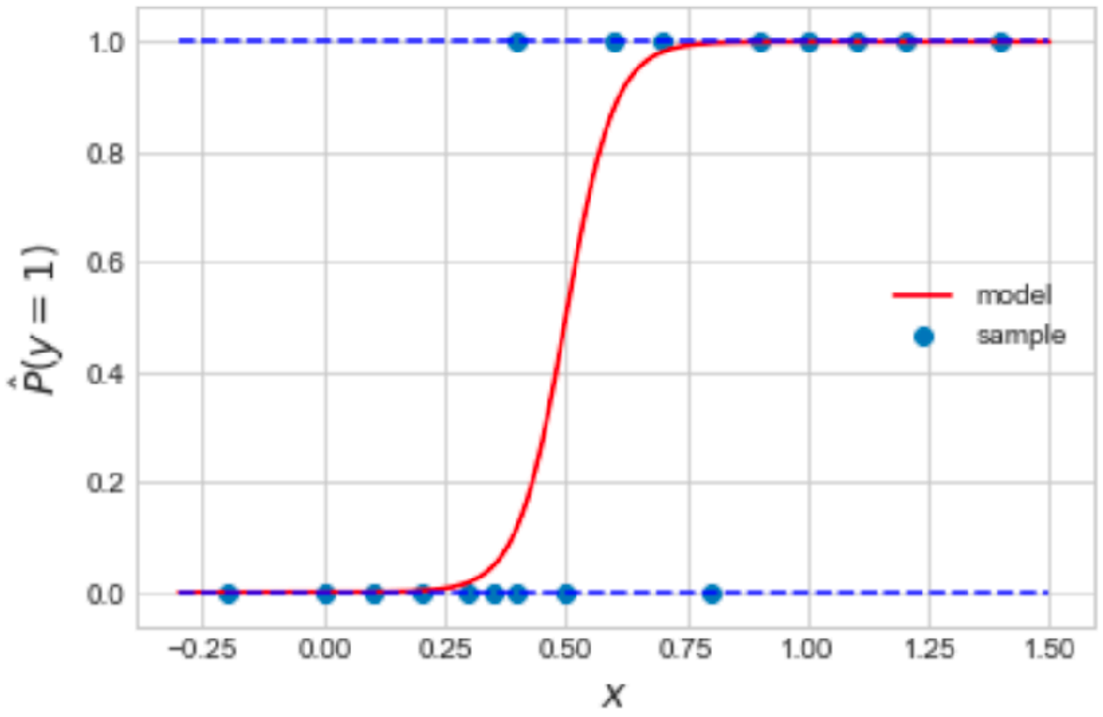
Классификация



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

Классификация

- $y \in \{0, 1\}$ — целевая переменная, X — признаки
- Модель: $P(y = 1 \mid w) = F\left(w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_k x_k\right)$



- В качестве $F(t)$ можно взять любую функцию распределения
- Если взять сигмоиду, модель будет интерпретируемая

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

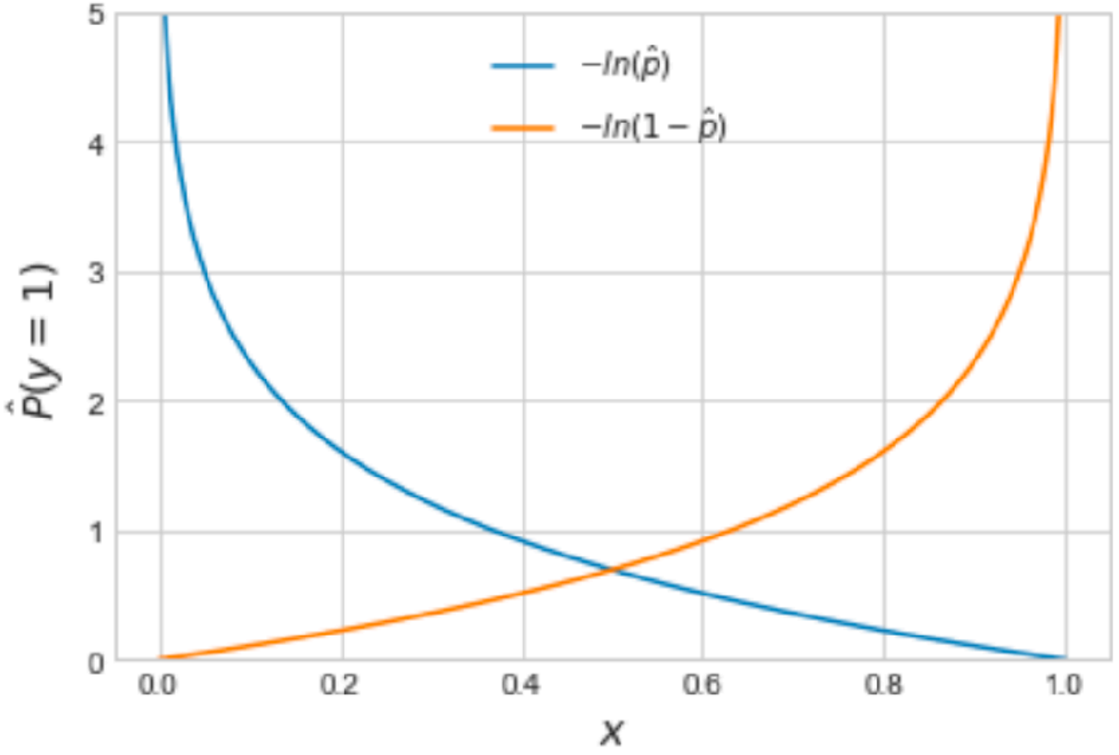
$$\sigma'(t) = \sigma(t) \cdot (1 - \sigma(t))$$

Функция потерь

- Наши y принимают значения 0 и 1

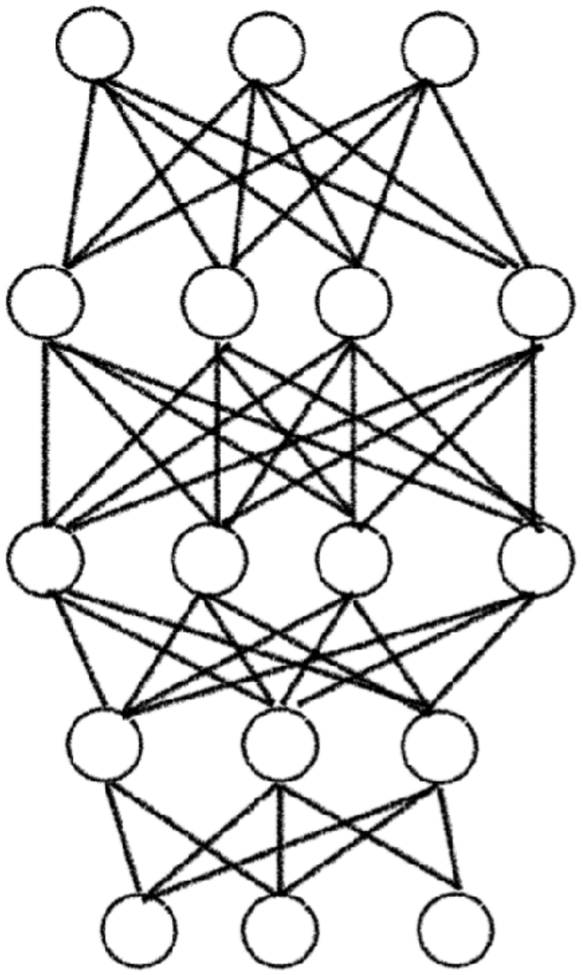
- Если $y = 1$, хотим большое $\hat{p} = \hat{P}(y = 1)$, но чем ближе $\hat{p}$ к 1, тем меньше хотим его увеличить

- Если $y = 0$, хотим большое $(1 - \hat{p})$, получается функция потерь:



$$\text{logloss} = -\frac{1}{n}\sum_{i=1}^{n} y_i \cdot \ln \hat{p} + (1 - y_i) \cdot \ln(1 - \hat{p})$$

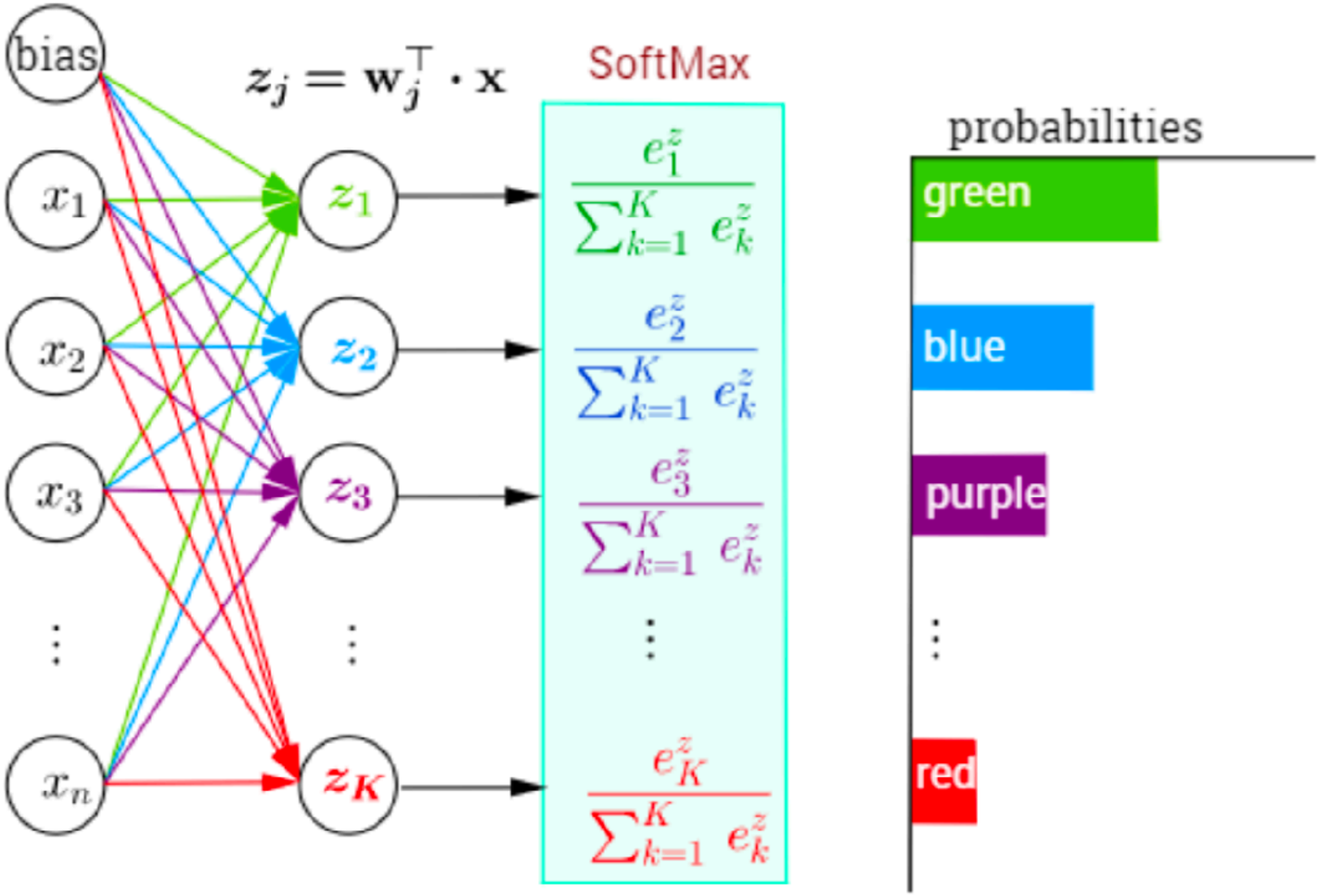
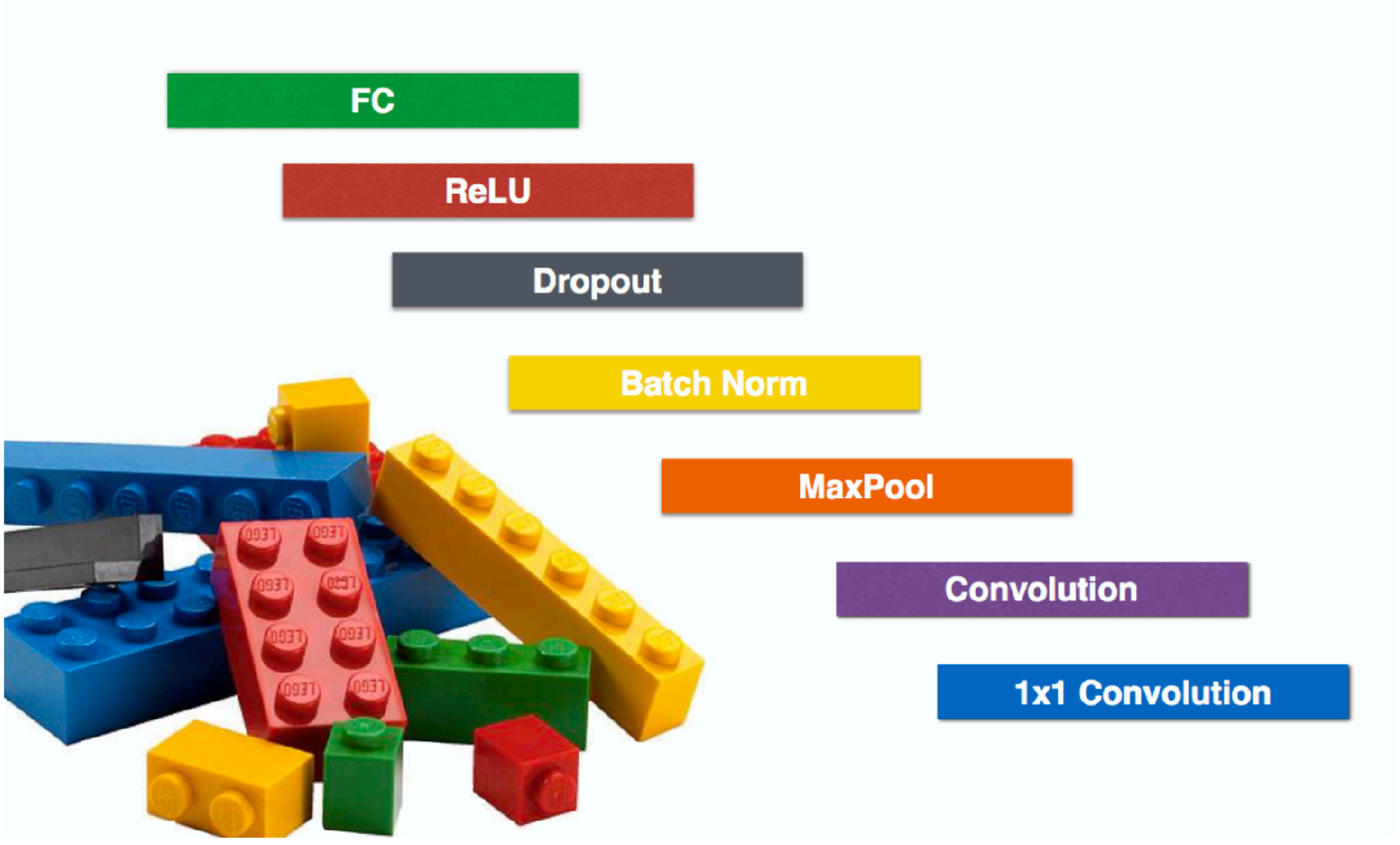
Мультиклассификация



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	$softmax(x)$
Output	

Мультиклассификация

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



Нейросети - конструктор LEGO