ВВЕДЕНИЕ В ГЛУБОКОЕ ОБУЧЕНИЕ

Сергей Николенко

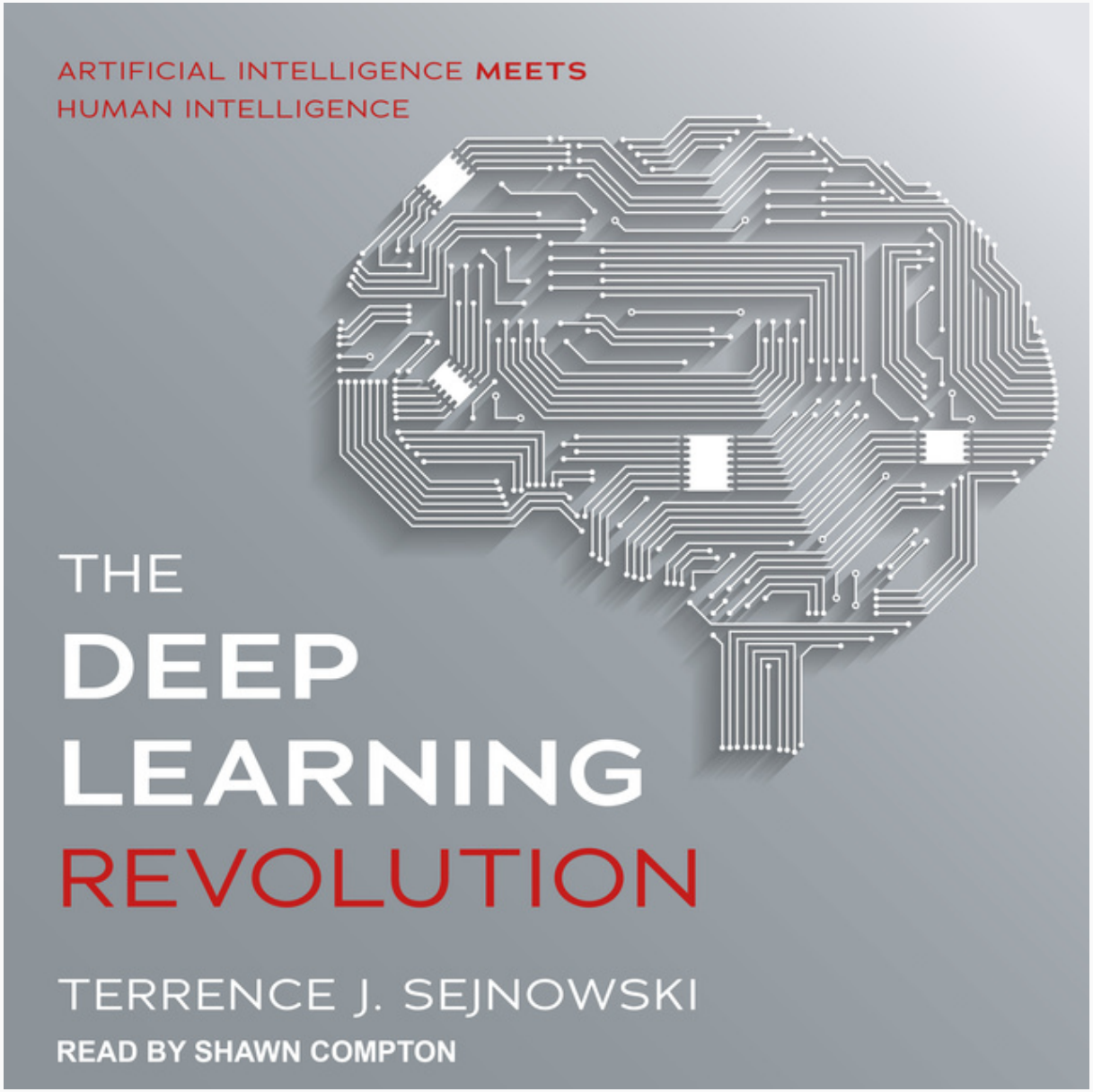
СПбГУ — Санкт-Петербург

05 сентября 2024 г.

Random facts:

- 5 сентября 1666 г. закончился Великий лондонский пожар, продолжавшийся 3 дня; сгорело около 10 тысяч зданий, но при этом известно всего о 16 погибших
- 5 сентября 1940 г. в «Пионерской правде» начали печатать повесть Аркадия Гайдара «Тимур и его команда», а 5 сентября 1958 г. в США был впервые опубликован роман Бориса Пастернака «Доктор Живаго»
- 5 сентября 1945 г., через три дня после конца Второй мировой войны, шифровальщик посольства СССР в Канаде Игорь Гузенко перешёл на сторону Канады, передав ей шифры и документы; считается, что этот скандал положил начало холодной войне
- 5 сентября 1964 г. группа «The Animals» заняла первое место в американском хит-параде со своей версией песни «The House of the Rising Sun»
- 5 сентября 1972 г. палестинская террористическая организация «Чёрный сентябрь» захватила команду Израиля во время Олимпийских игр в Мюнхене
- 5 сентября 1997 г. в Москве был открыт памятник Петру I работы Зураба Церетели
- 5 сентября 1998 г. Ким Ир Сен был признан вечным президентом КНДР

Революция глубокого обучения



План

- Что же такое глубокие нейронные сети, как они работают и почему так хороши?
- Наш план:
  (1) начнём с того, что же такое в мозге интересное происходит,
  (2) затем перейдём к нейронным сетям, обсудим, как они работают,
  (3) изучим много разных архитектур нейронных сетей, особенно свёрточных на примере распознавания объектов и сегментации,
  (4) поговорим о механизмах внимания и особенно трансформерах
  (5) и закончим разговором о глубоких порождающих моделях.

Краткое резюме предыдущих серий

- В машинном обучении много разных задач, с учителем и без.
- Их обычно решают по теореме Байеса, пересчитывая наши представления о параметрах в зависимости от полученных данных.
- Для этого обычно надо оптимизировать какую-нибудь сложную функцию (например, апостериорное распределение).
- И для невыпуклых функций это обычно делают тем или иным вариантом градиентного спуска.
- А теперь – о том, при чём тут нейроны и сети.

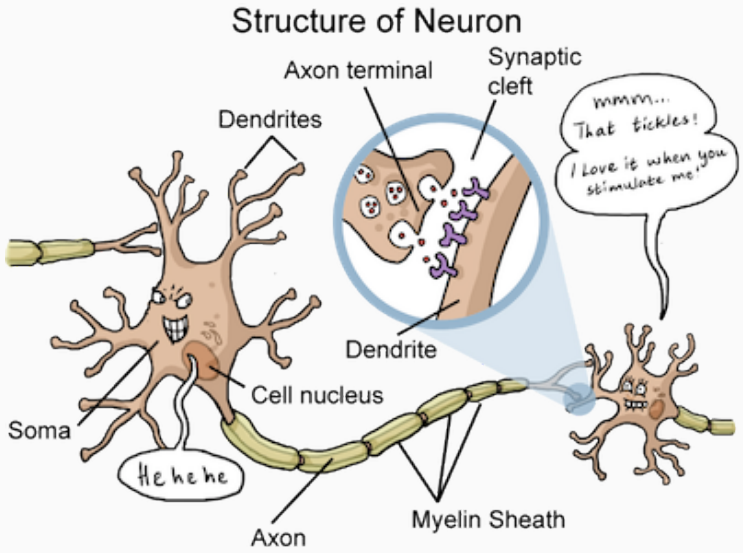
Человеческий мозг и история искусственных нейронных сетей

Компьютер у нас в голове

- Компьютер считает быстрее человека, но гораздо хуже
  - понимает естественный язык,
  - распознаёт изображения, узнаёт людей (это уже не совсем так),
  - обучается в широком смысле этого слова и т.д...
- Почему так? Как человек всего этого добивается?

Компьютер у нас в голове

- В мозге много нейронов; каждый нейрон:
  - через дендриты получает сигнал от других нейронов;
  - время от времени запускает сигнал по аксону;
  - через синапсы сигнал аксона доходит до дендритов других нейронов.

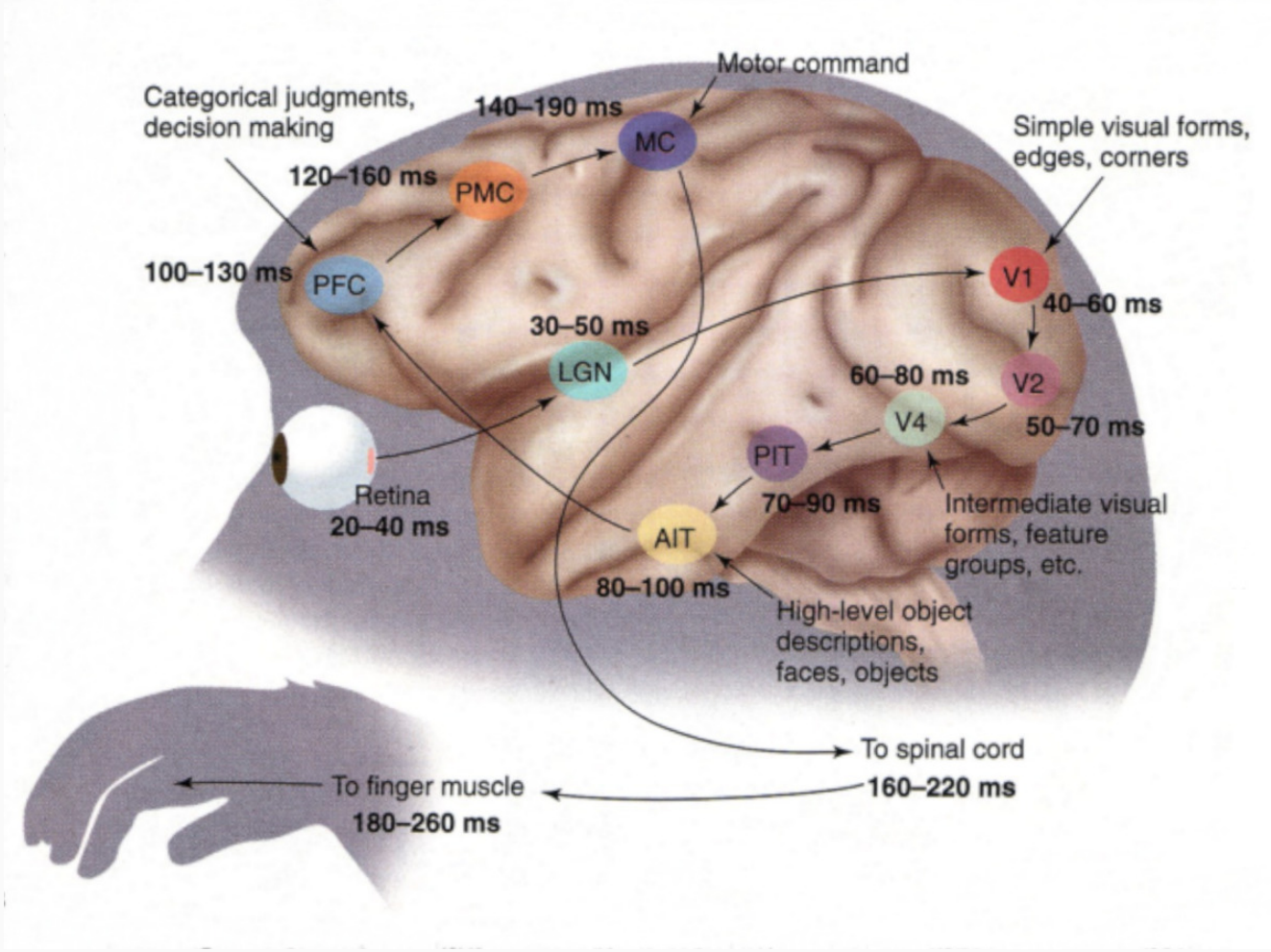


Компьютер у нас в голове

- Нейрон время от времени, стохастически, выдаёт сигналы.
- Всего нейронов очень много: 10^{11} нейронов, в среднем 7000 связей у каждого, т.е. 10^{15} синапсов.
- Но firing rate от 10 до 200 герц.
- А распознаём мы лицо за пару сотен миллисекунд.
- Это значит, что цепочки короткие, и вычисления скорее параллельные, чем последовательные.

Компьютер у нас в голове

- Вот как мы обрабатываем картинку:



Обучение признаков

- Другая сторона вопроса – обучение признаков.
- Мозг очень хорошо умеет обучаться на очень-очень маленькой выборке данных.
- И может адаптироваться к новым источникам информации.
- Как он это делает? Можем ли мы сделать так же?

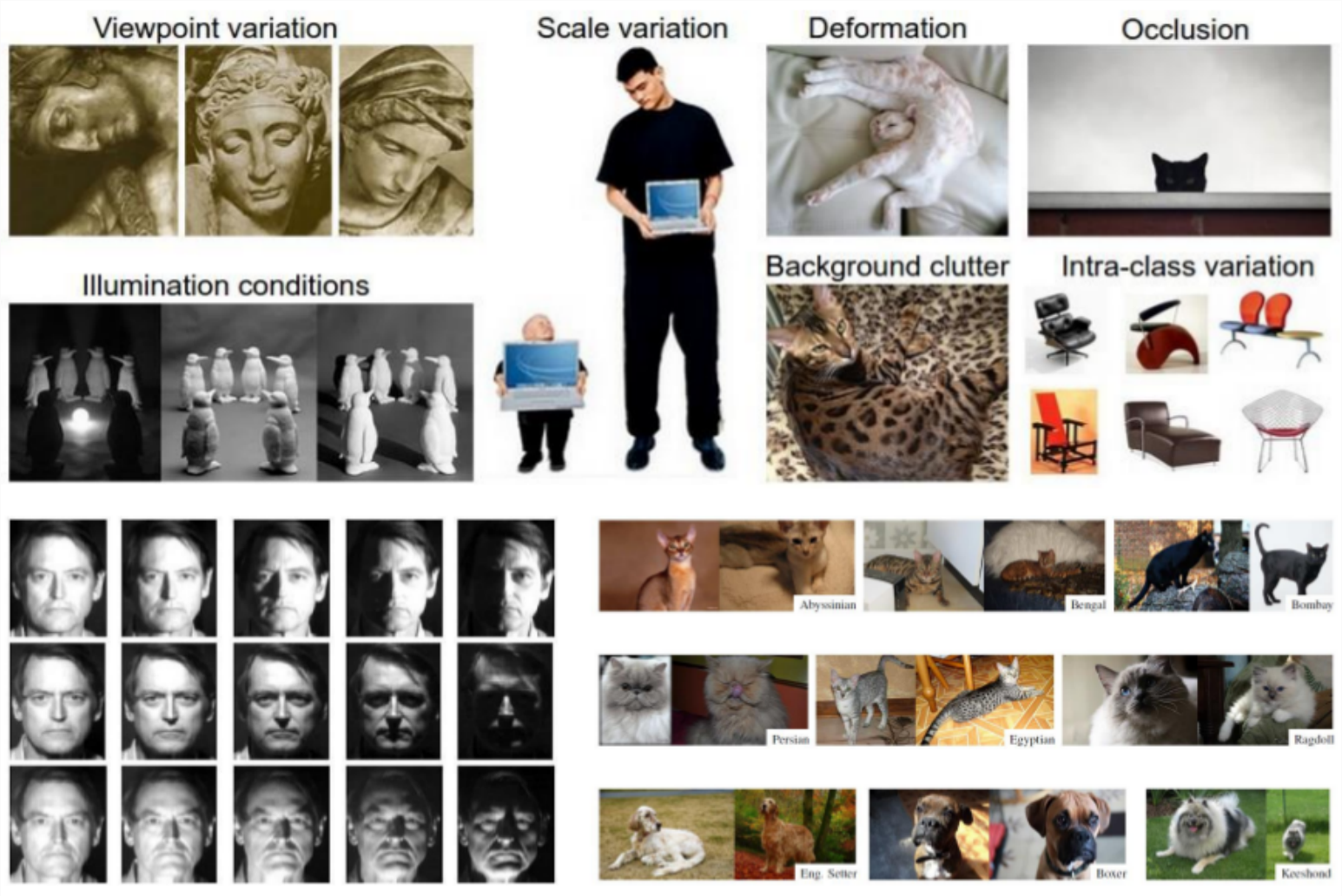
Обучение признаков

- Системы обработки неструктурированной информации выглядят обычно так:

$$\text{вход} \to \text{признаки} \to \text{классификатор}$$

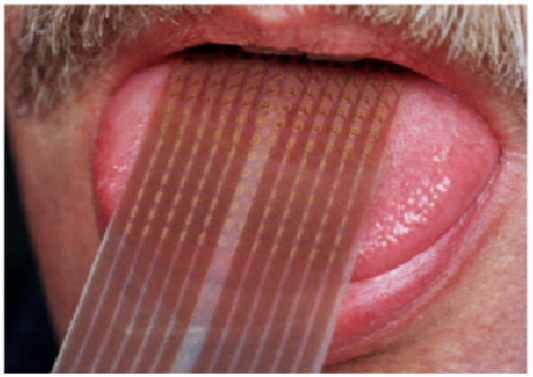
- Люди много десятилетий пытались придумать хорошие признаки:
  - MFCC для распознавания речи;
  - SIFT для обработки изображений;
  - ...
- Задача feature engineering: как сделать такие признаки?
- Но, может быть, можно найти признаки автоматически? Мозг ведь это как-то делает...
- Но это сложно!

Почему это сложно: распознавание изображений



Пластичность

- Третий аспект — пластичность мозга.



- Она показывает, что есть единый алгоритм обучения, который можно применять к самым разным ситуациям.

Предостережение

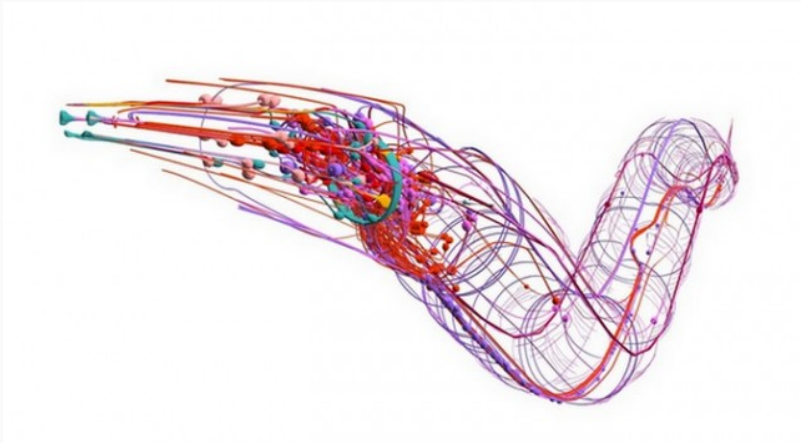
- Мы ещё не раз вернёмся к тому, как работает настоящий мозг и настоящие нейроны.
- Но лучше не слишком увлекаться:
  - 2004 – может ли биолог починить радиоприёмник?

  - 2016 – может ли нейробиолог понять микропроцессор?

Предостережение

- Мало что понимаем в целых нервных системах:
  - Human Brain Project Генри Маркрама провалился;

  - а получается пока OpenWorm
    (нематода *C. elegans*, 302 нейрона).



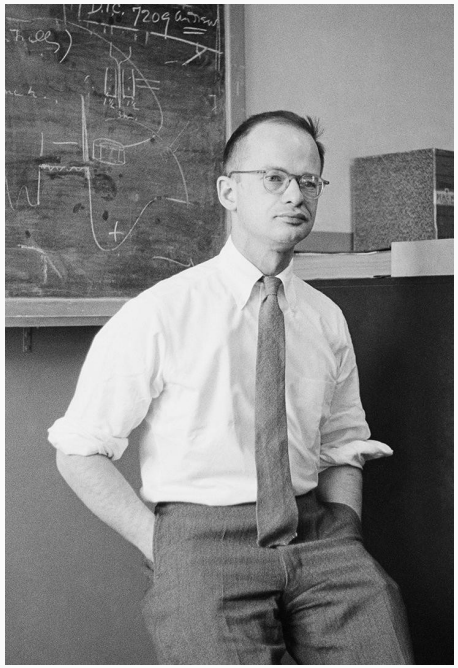
- Так что будем вдохновляться, но без фанатизма.

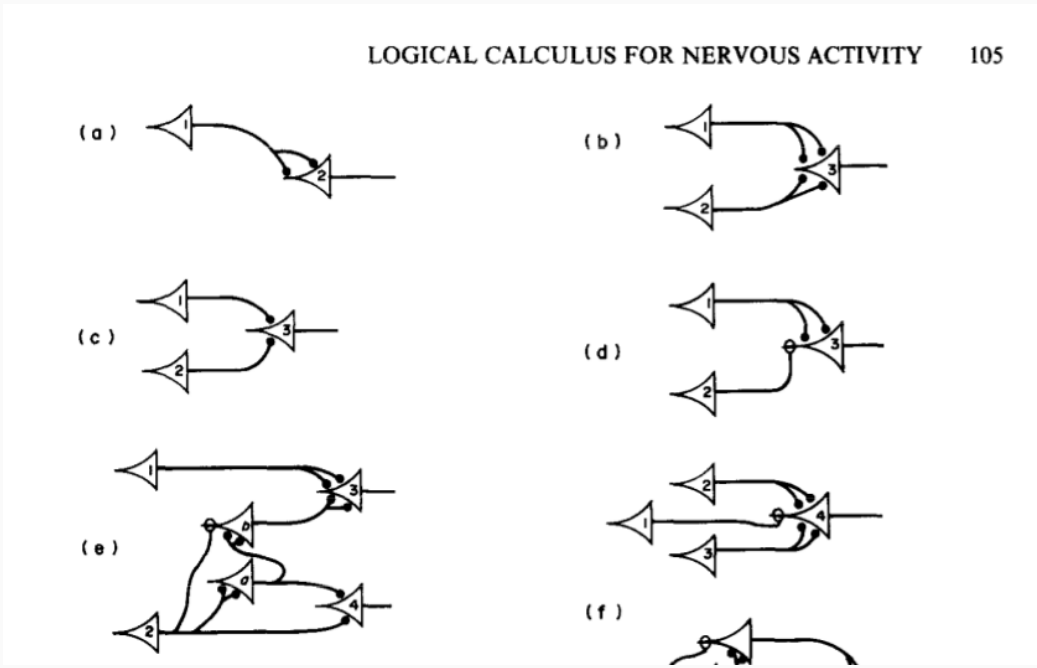
Суть происходящего

- Вообще говоря, всё, что мы делаем в машинном обучении, — это аппроксимация и оптимизация функций.
- Например:
  - есть функция из картинок в то, что на них изображено;
  - ещё проще: бинарная функция из пикселей картинки в «котик или не котик»;
  - функция, мягко скажем, довольно сложная; задана в виде датасета; как нам её реализовать?
- Обычный подход в машинном обучении:
  - строим какой-то класс моделей, обычно параметрический;
  - и подгоняем параметры так, чтобы модель хорошо описывала данные;
  - то есть оптимизируем какую-то функцию ошибки или функцию качества (правдоподобие, апостериорную вероятность).
- Нейронные сети — это очень мощный и гибкий класс таких моделей.
- Сложность в том, как же обучить их параметры.

История ANN и перцептрон

- Давайте попробуем как в природе: есть связанные между собой нейроны, которые передают друг другу сигналы.
- McCulloch, Pitts, 1943: идея.

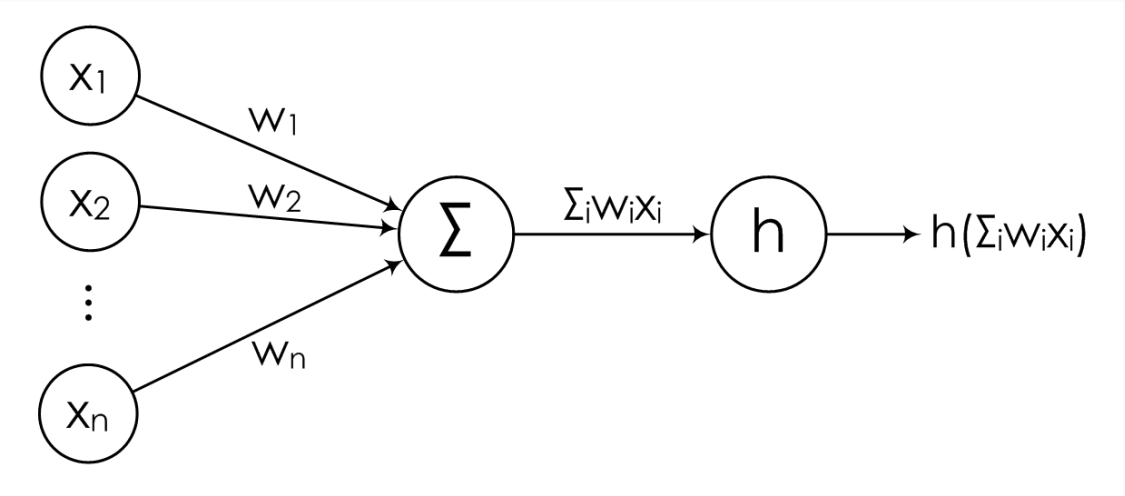




- Идею искусственных сетей, похожих на современные (даже с несколькими уровнями), предлагал ещё Алан Тьюринг (1948).
- Rosenblatt, 1958: перцептрон (один искусственный нейрон).
- Тогда же появился алгоритм обучения градиентным спуском.

История ANN и перцептрон

- Один нейрон обычно моделируется вот так:



- Линейная комбинация входов, потом нелинейность:

$$y = h(\mathbf{w}^\top \mathbf{x}) = h\left(\sum_i w_i x_i\right).$$

- Как обучить перцептрон, и каков будет результат?

История ANN и перцептрон

- Обучение *градиентным спуском*.
- Для исходного перцептрона Розенблатта ($h = \mathrm{id}$):

$$E_P(\mathbf{w}) = -\sum_{\mathbf{x}\in\mathcal{M}} y(\mathbf{x})\left(\mathbf{w}^\top \mathbf{x}\right),$$

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla_{\mathbf{w}} E_P(\mathbf{w}) = \mathbf{w}^{(\tau)} + \eta t_n \mathbf{x}_n.$$

- Или, к примеру, можно делать классификацию, подставляя в качестве функции активации логистический сигмоид $h(x) = \sigma(x) = \frac{1}{1+e^{-x}}$:

$$E(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log\sigma(\mathbf{w}^\top \mathbf{x}_i) + (1-y_i)\log\left(1-\sigma(\mathbf{w}^\top \mathbf{x}_i)\right)\right).$$

- По сути это просто логистическая регрессия.

История ANN и перцептрон

- Первый перцептрон (Rosenblatt, 1958) распознавал цифры.

История ANN и перцептрон

- И люди в него верили:

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human beings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

1958 New York Times...

In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

Learns by Doing

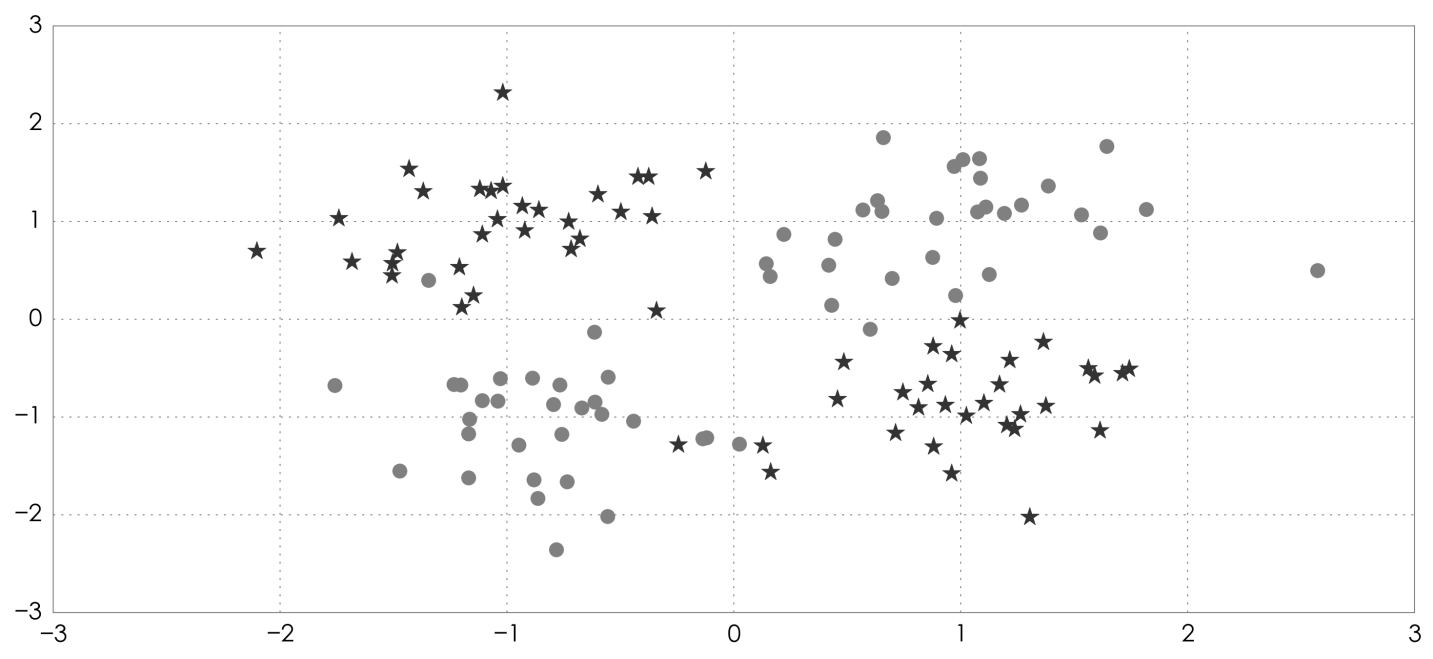
In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.

История ANN и перцептрон

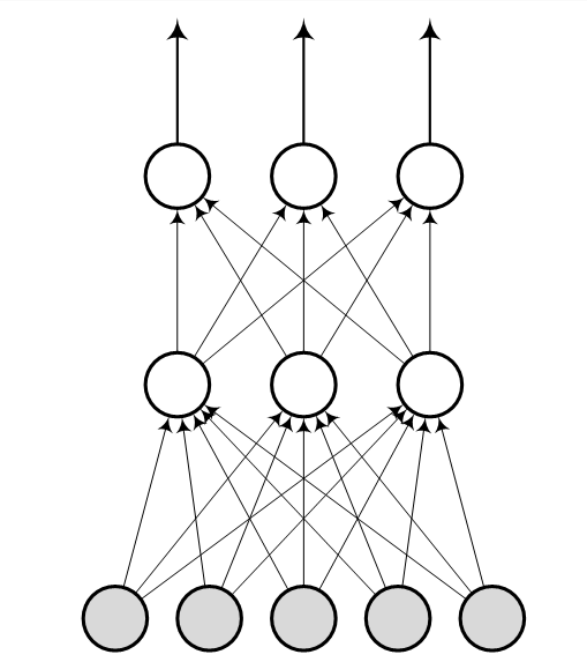
- 1960-е годы: изучали перцептроны.
- (Minsky, Papert, 1969): даже с нелинейностью один перцептрон задаёт только линейную разделяющую поверхность, XOR нельзя моделировать перцептроном...



- Это почему-то восприняли как большую проблему, которая ставит крест на нейронных сетях.

История ANN и перцептрон

- ...хм, ну конечно! Для нелинейных функций нужна сеть перцептронов:



История ANN и перцептрон

- (Brison, Hoback, 1969): предложили алгоритм обратного распространения ошибки (backpropagation).
- (Hinton, 1974): переоткрыл backpropagation, с тех пор он стал популярным.
- Во второй половине 1970-х появились многоуровневые ANN, была разработана современная теория.
- Глубокие модели появились в первой половине 1980-х гг.! Это вообще не очень хитрая идея сама по себе.

История ANN и перцептрон

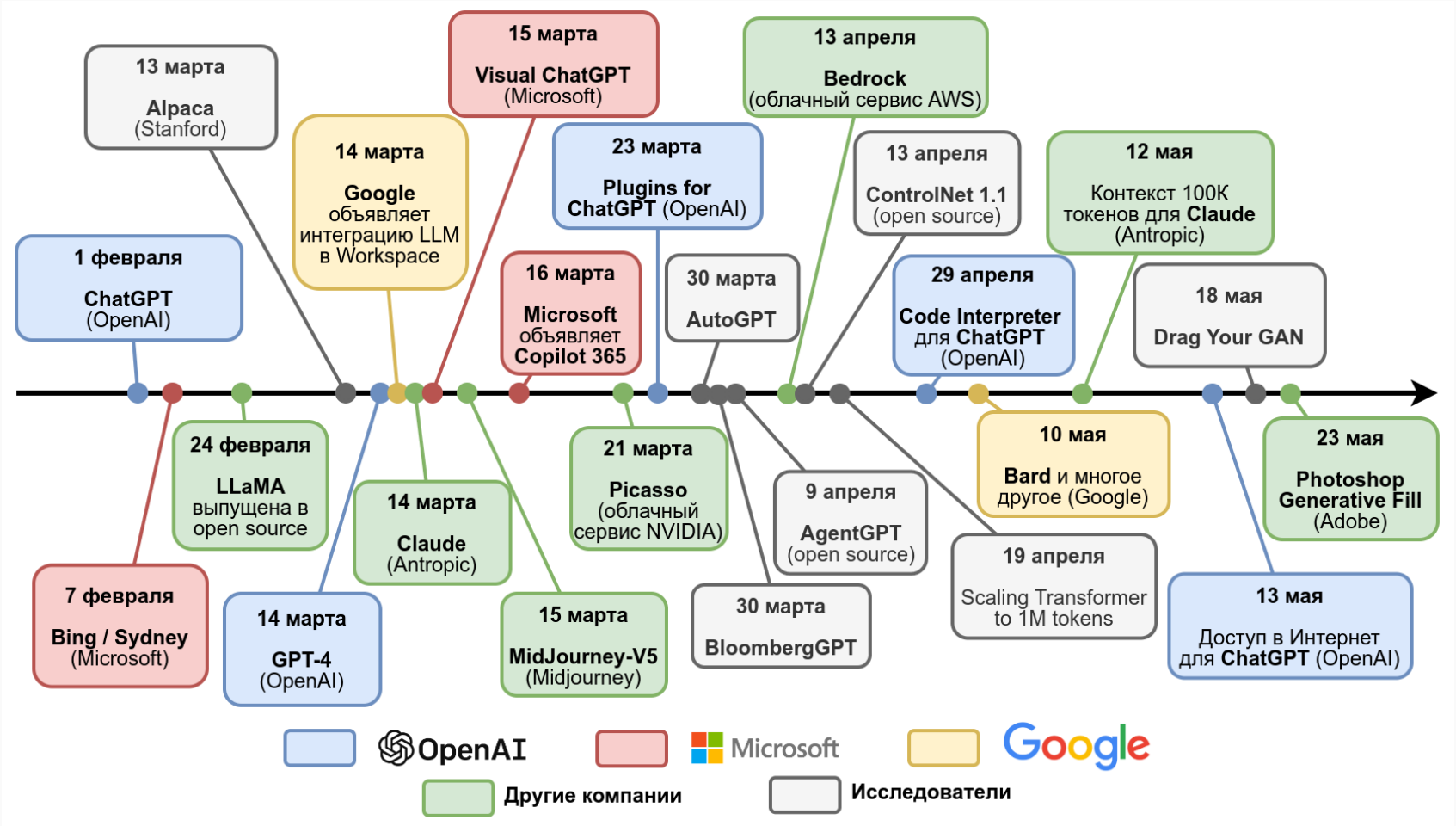
- Но к началу 1990-х решили, что нейронные сети использовать смысла нет, и началась «вторая зима».
- Это было потому, что тогда ни математически, ни вычислительно не могли нормально обучить большие модели.
- Нужно было обучать сети неделями, а качество, например, на распознавании речи проигрывало HMM, в других задачах проигрывало другим методам.
- John Denker, 1994: «neural networks are the second best way of doing just about anything».
- К концу 90-х на нейросетях остались только обработка изображений (Yann LeCun) и несколько других групп (Geoffrey Hinton, Yoshua Bengio).

История ANN и перцептрон

- Революция deep learning начинается в 2006: в группе Джеффри Хинтона изобрели Deep Belief Networks (DBN).
- Основная идея: научились делать *обучение без учителя*, выделять признаки (при помощи машин Больцмана, RBM; появились в 1983), использовать это как предобучение.
- Компьютеры стали мощнее, появились способы передать вычисления на GPU, и вычислительно мы смогли обрабатывать гораздо более объёмные датасеты, которые как раз стали доступными.
- А потом появились новые методы регуляризации (дропаут, затем batch normalization), и RBM с предобучением тоже стали не очень нужны.

История ANN и перцептрон

- Сейчас мы живём во время очередной мини-революции, на этот раз языковых моделей.



История ANN и перцептрон

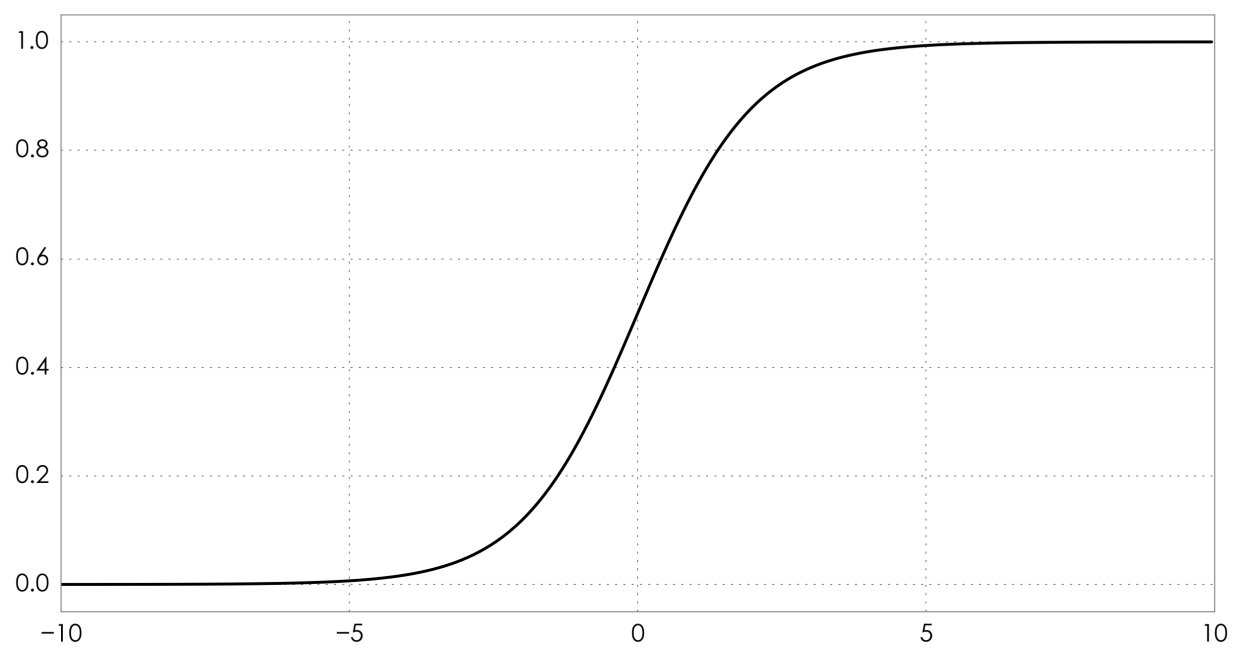
- Многие считают, что AGI не за горами!
- Вот основные компоненты революции глубокого обучения.
- Осталось только понять, что всё это значит. :)
- Но сначала давайте до конца разберёмся с одним перцептроном.

Один перцептрон: функции активации

Функции активации

- Есть много разных нелинейных функций, которые можно использовать в перцептроне.
- Логистический сигмоид:

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

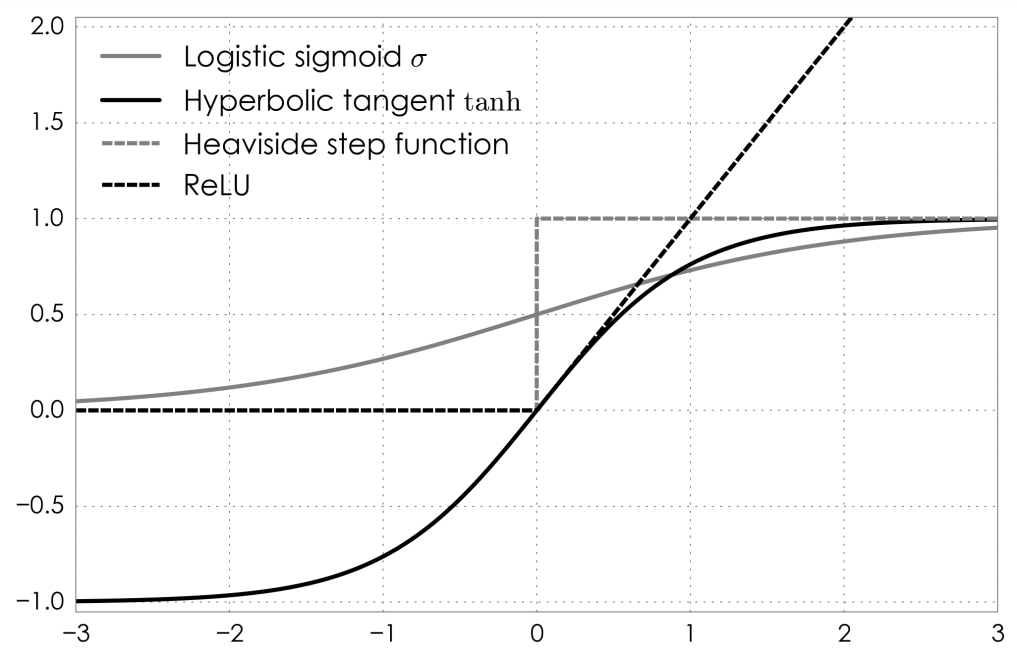


Функции активации

- Гиперболический тангенс:

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

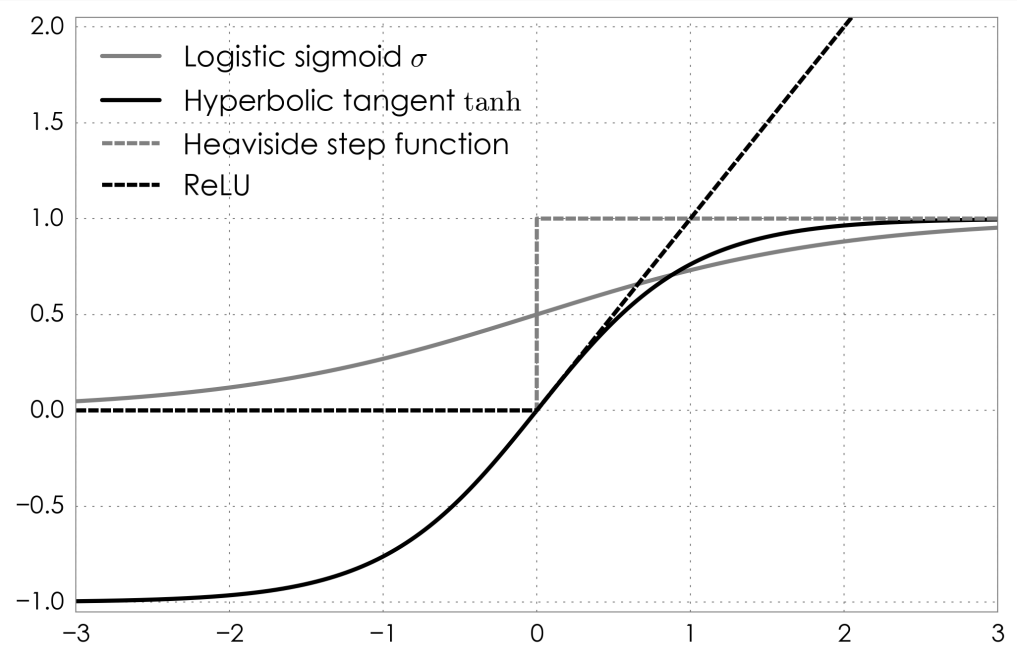
- Очень похоже, но ноль не является точкой насыщения.



Функции активации

- Ступенька (функция Хевисайда):

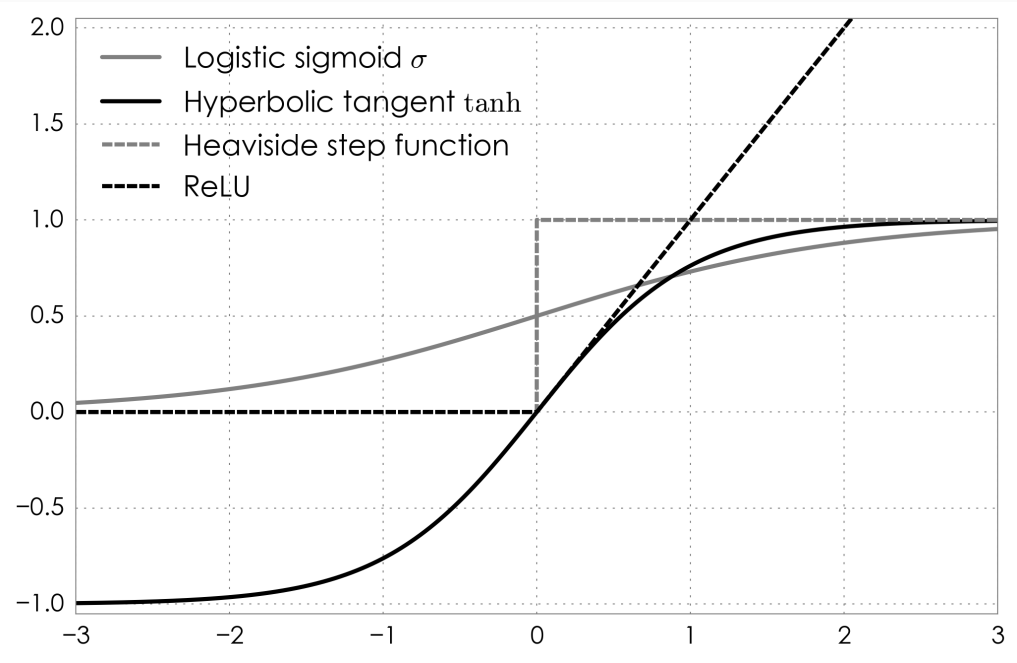
$$\text{step}(x) = \begin{cases} 0, & \text{if } x < 0, \\ 1, & \text{if } x > 0. \end{cases}$$



Функции активации

- Rectified linear unit (ReLU):

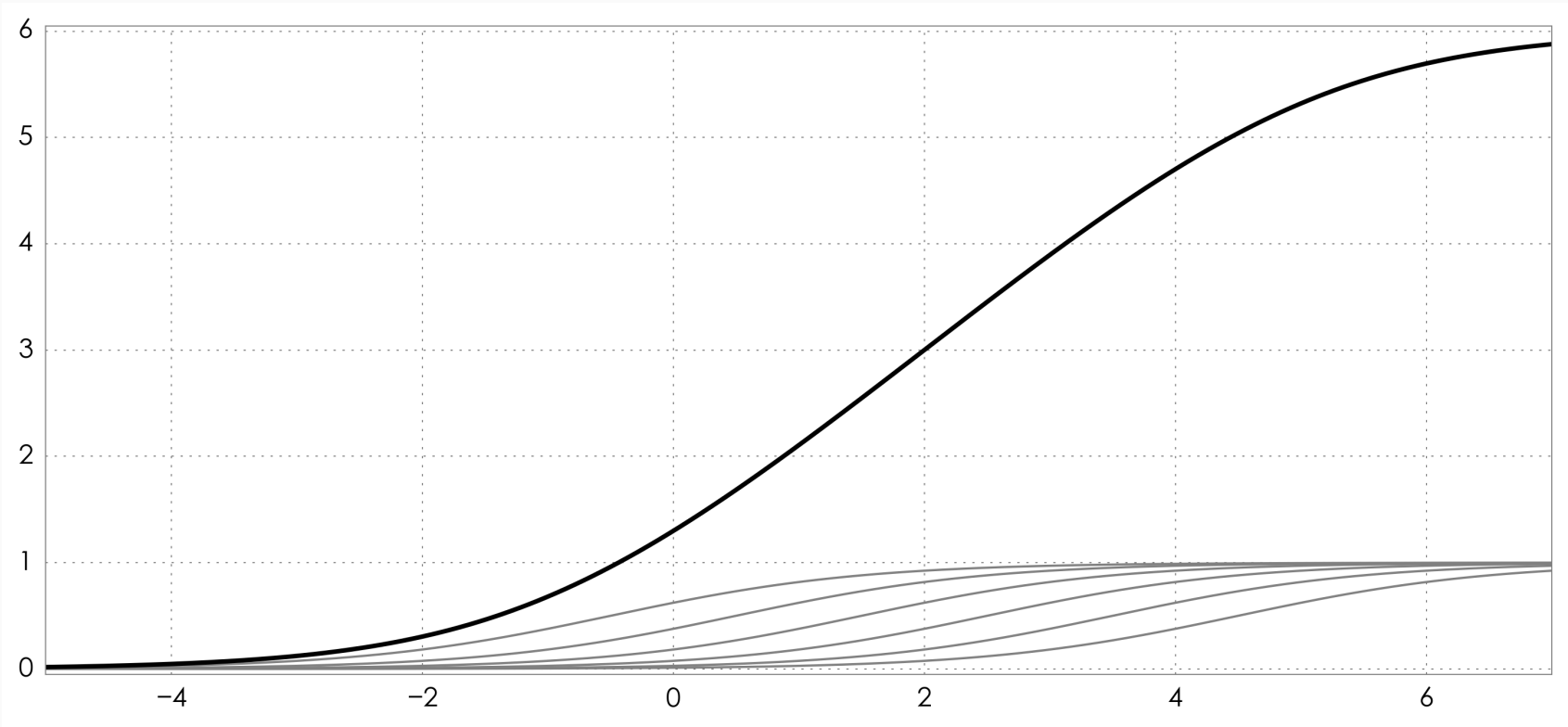
$$\mathrm{ReLU}(x) = \begin{cases} 0, & \text{if } x < 0, \\ x, & \text{if } x \geq 0. \end{cases}$$



Функции активации

- Математическая мотивация для ReLU:

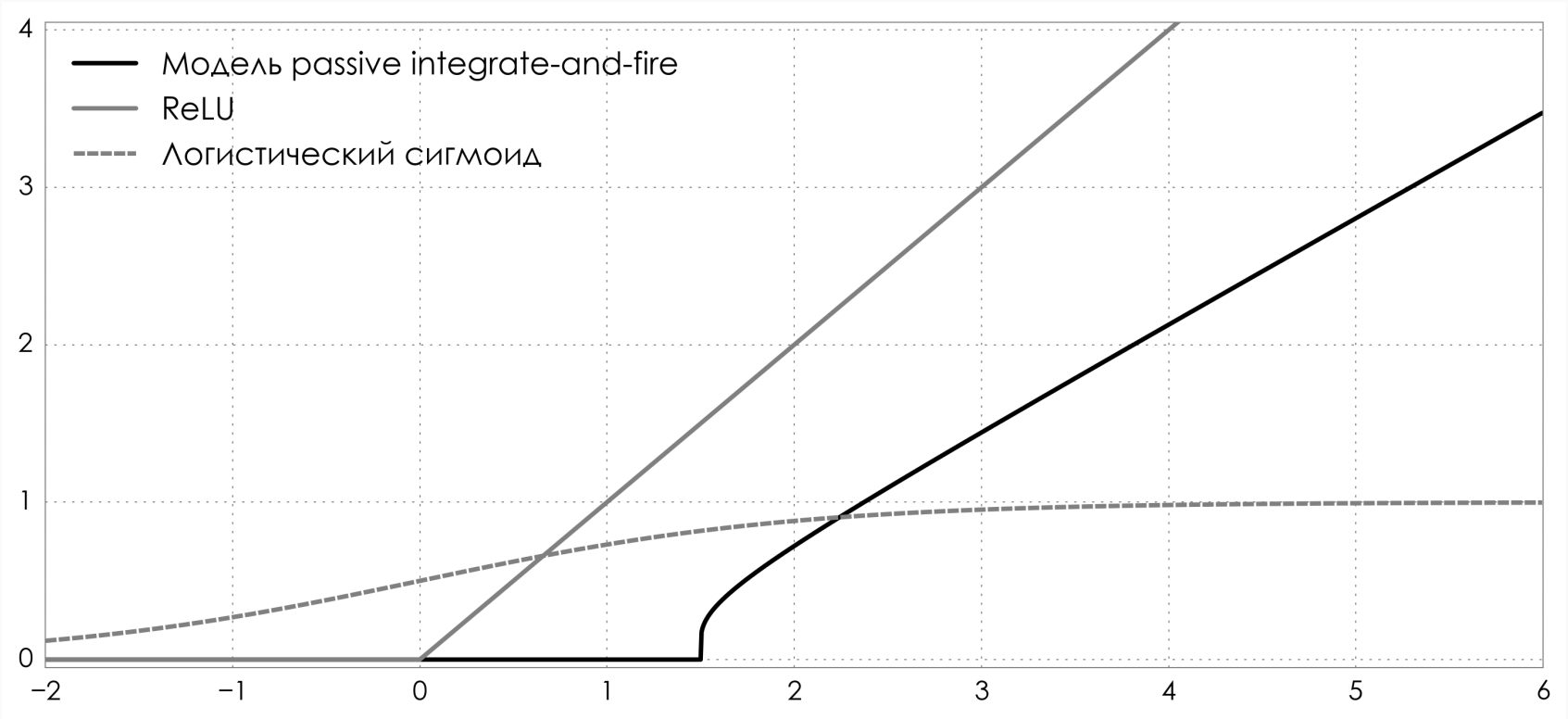
$$\sigma\left(x+\frac{1}{2}\right)+\sigma\left(x-\frac{1}{2}\right)+\sigma\left(x-\frac{3}{2}\right)+\sigma\left(x-\frac{5}{2}\right)+\ldots.$$



Функции активации

- Биологическая мотивация для ReLU:

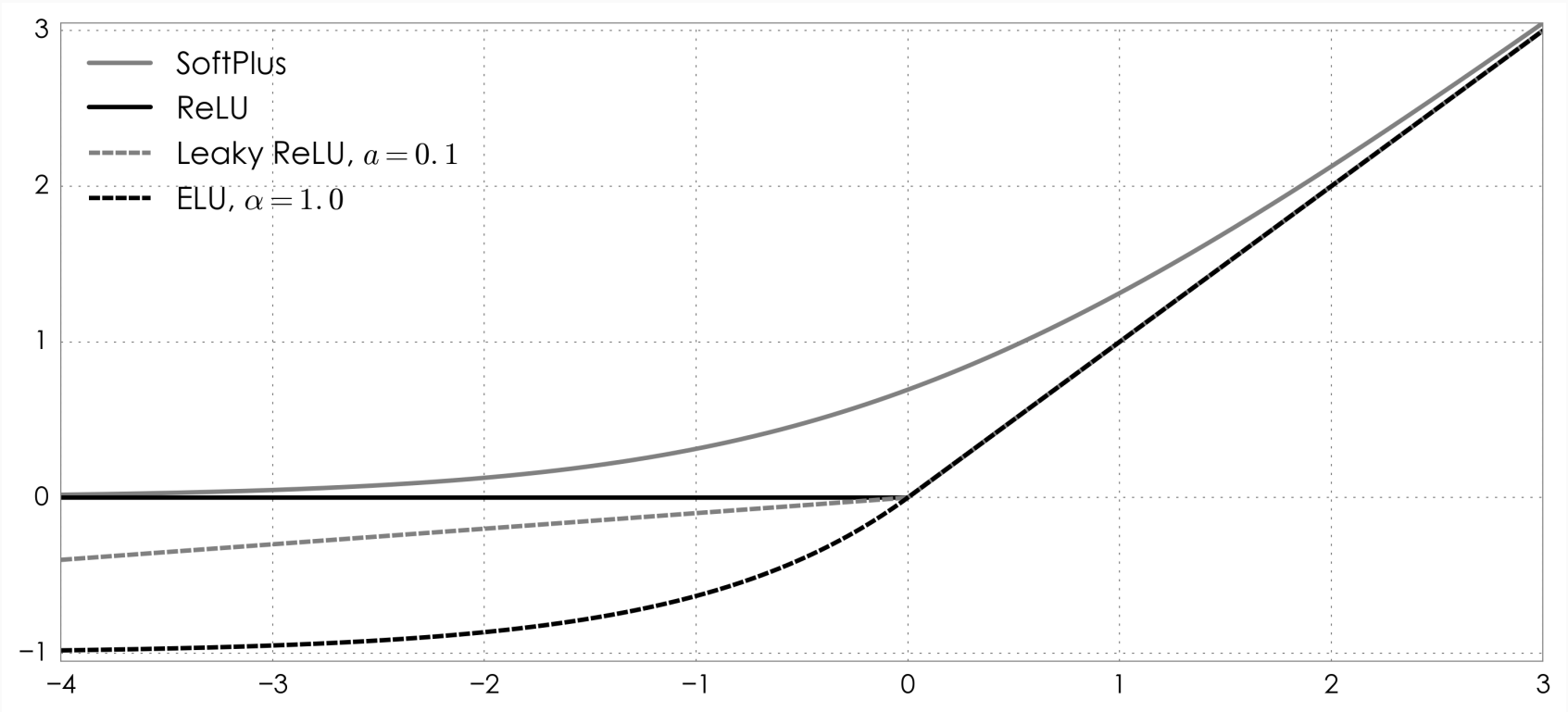
$$f(I) = \begin{cases} \left(\tau \log \frac{E+RI-V_{\text{reset}}}{E+RI-V_{th}}\right)^{-1}, & \text{if } E + RI > V_{th}, \\ 0, & \text{if } E + RI \leq V_{th}, \end{cases}$$



Функции активации

- Несколько вариантов ReLU — Leaky ReLU и Parametric ReLU:

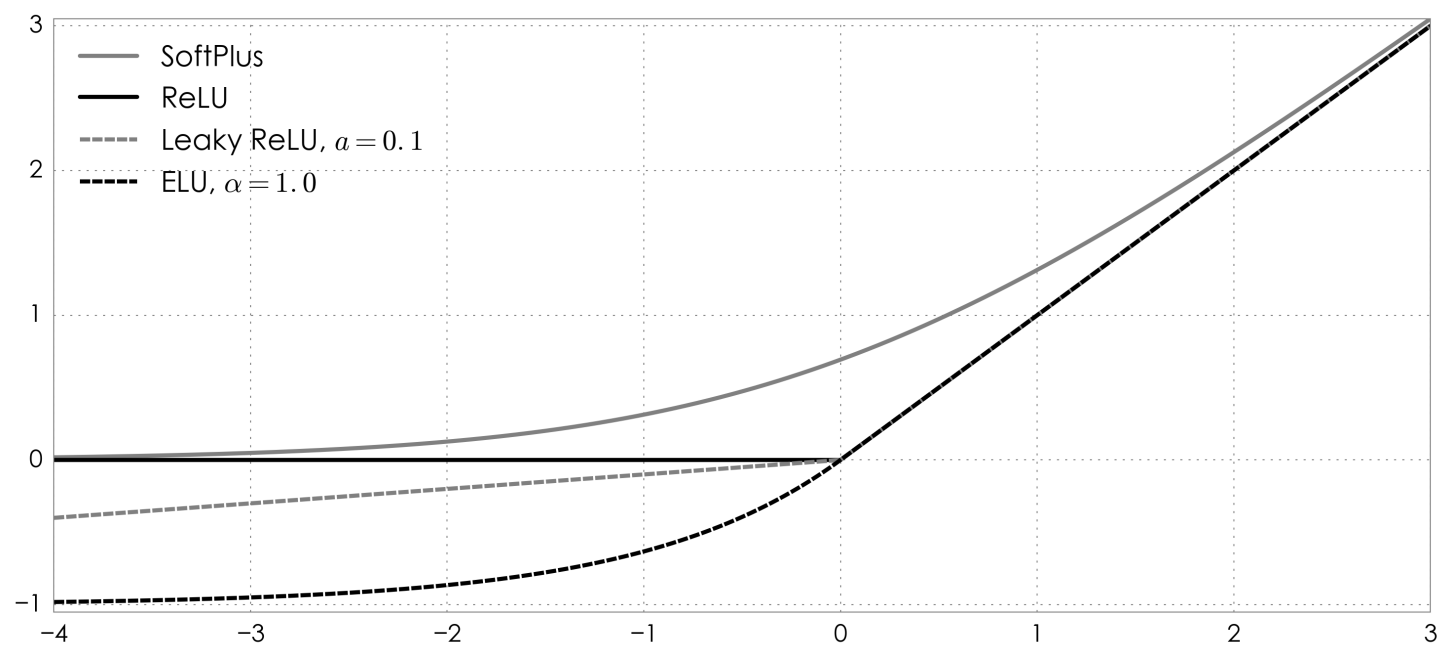
$$\mathrm{LReLU}(x) = \begin{cases} ax, & \text{if } x < 0, \\ x, & \text{if } x > 0. \end{cases}.$$



Функции активации

- Exponential LU:

$$\mathrm{ELU}(x) = \begin{cases} \alpha\left(e^x - 1\right), & x < 0, \\ x, & x \geq 0. \end{cases}$$



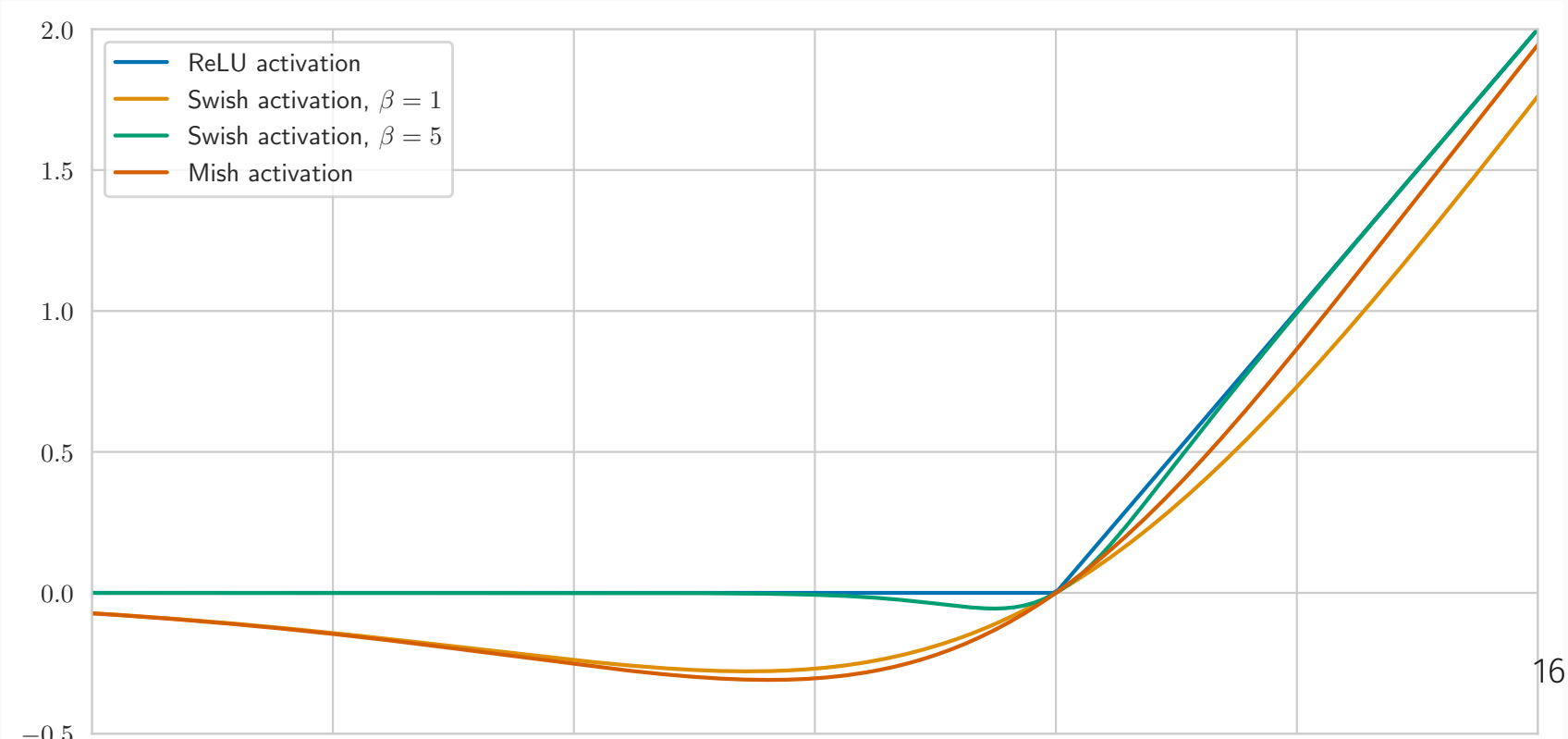
- Обычно ReLU хорошо работает, но последнего слова ещё не сказано.

Функции активации

- Но история только начинается. Ramachandran et al. (2017) сделали поиск по пространству функций и нашли *Swish*:

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

- (Misra, 2019): $\text{Mish}(x) = x \tanh(\text{softplus}(x)) = x \tanh(\ln(1 + e^x))$



Функции активации

- Очень интересная работа вышла в сентябре 2020 года...
- (Ma et al., 2020): давайте рассмотрим гладкое обобщение максимума

$$S_\beta(x_1, \dots, x_n) = \frac{\sum_{i=1}^n x_i e^{\beta x_i}}{\sum_{i=1}^n e^{\beta x_i}}.$$

- И давайте подставим туда две функции от x, т.е. жёсткому максимуму $\max(g(x), h(x))$ будет соответствовать

$$\begin{aligned} S_\beta(g(x), h(x)) &= g(x)\frac{e^{\beta g(x)}}{e^{\beta g(x)} + e^{\beta h(x)}} + h(x)\frac{e^{\beta h(x)}}{e^{\beta g(x)} + e^{\beta h(x)}} = \\ &= g(x)\sigma(\beta(g(x) - h(x))) + h(x)\sigma(\beta(h(x) - g(x))) = \\ &= (g(x) - h(x))\,\sigma(\beta(g(x) - h(x))) + h(x). \end{aligned}$$

Функции активации

- Ma et al. назвали это *ActivateOrNot* (ACON). Теперь это обобщает всё на свете:
  - для $g(x) = x$ и $h(x) = 0$ жёсткий максимум будет $\mathrm{ReLU}(x) = \max(x, 0)$, а гладкий максимум

$$f_{ACON-A}(x, 0) = S_\beta(x, 0) = x\sigma(\beta x),$$

    т.е. в точности *Swish*!
  - для $g(x) = x$ и $h(x) = ax$ при некотором $a < 1$, жёсткий максимум будет $\mathrm{LReLU}(x) = \max(x, px)$, а гладкий максимум

$$f_{ACON-B} = S_\beta(x, ax) = (1 - a)\, x\sigma(\beta(1 - a)x) + ax;$$

Функции активации

- Ma et al. назвали это *ActivateOrNot* (ACON). Теперь это обобщает всё на свете:
  - и это можно прямолинейно обобщить до

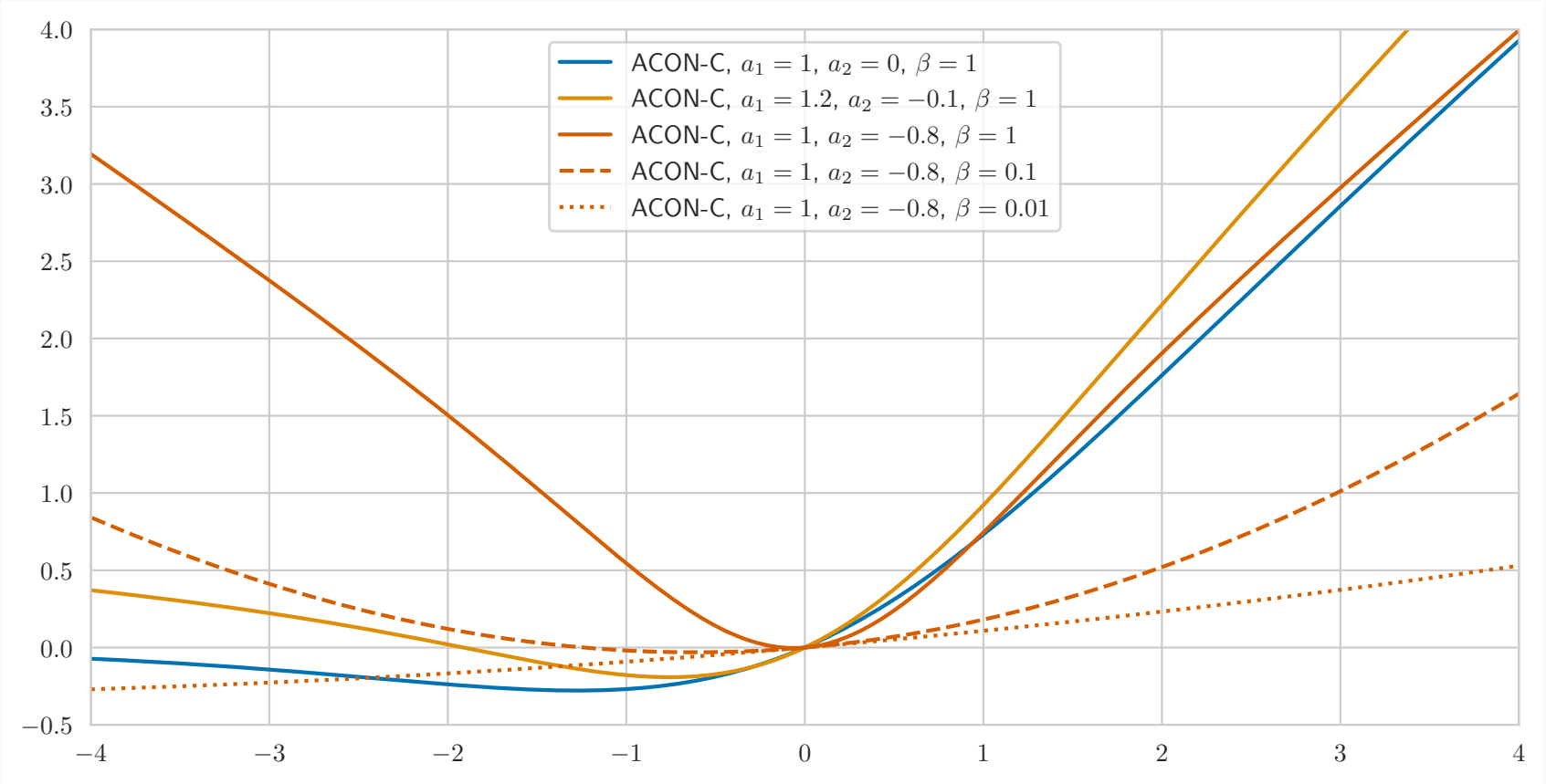
$$f_{ACON-C} = S_\beta(a_1 x, a_2 x) = (a_1 - a_2)\, x\sigma\left(\beta(a_1 - a_2)x\right) + a_2 x;$$

теперь a_1 и a_2 могут стать обучаемыми параметрами, а интуитивно это просто пределы производной с двух сторон:

$$\lim_{x\to\infty} \frac{\mathrm{d}f_{ACON-C}}{\mathrm{d}x} = a_1, \qquad \lim_{x\to-\infty} \frac{\mathrm{d}f_{ACON-C}}{\mathrm{d}x} = a_2.$$

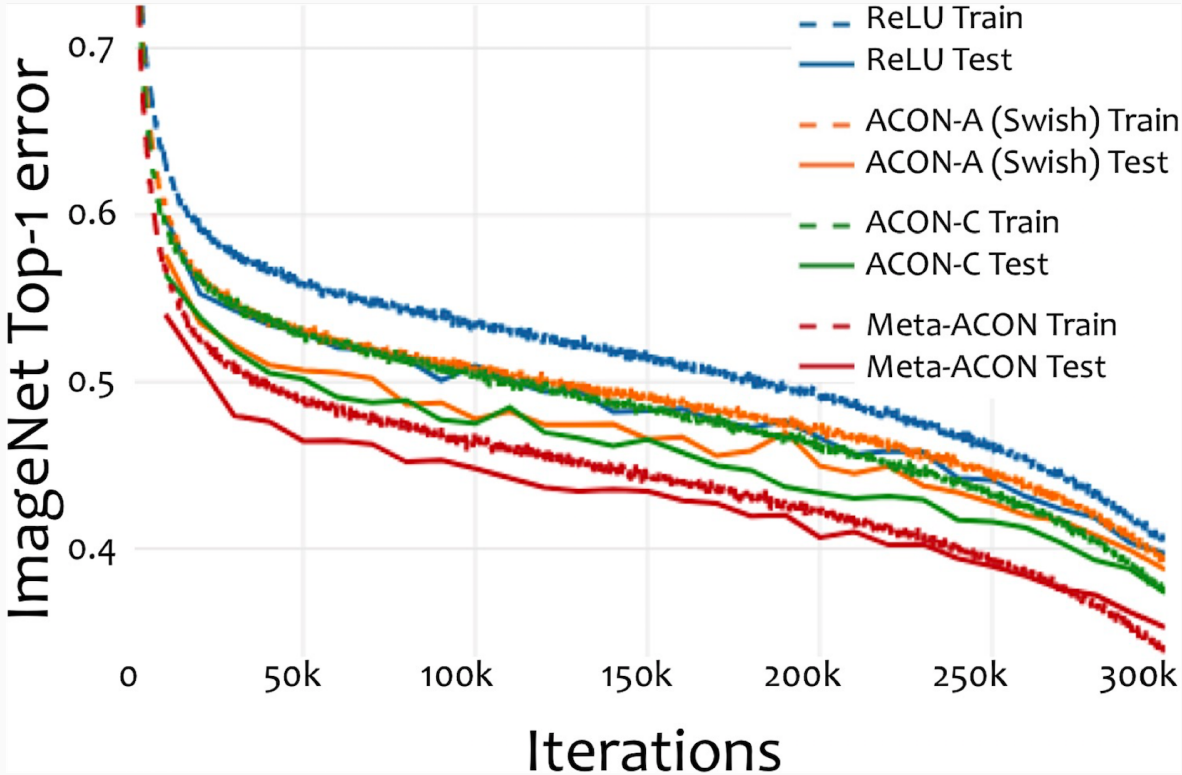
Функции активации

- Вот так выглядит ACON-C для разных значений параметров:



Функции активации

- Пишут, что подбором параметров можно очень много выгадать даже в самых стандартных избитых задачах:



- Но как-то с тех пор не произошло революции...
- Это частая история в DL.

Настоящие нейроны

- Есть ли градиентный спуск в настоящих нейронах?
- Скорее всего нет (увидим позже почему).
- Хотя у Хинтона есть интересный доклад об этом... но всё-таки вряд ли.
- *Обучение по Хеббу* – neurons that fire together, wire together:

$$\Delta w_i = \eta x_i x_j.$$

- В ML привело к *сетям Хопфилда*, которые вроде как реализуют ассоциативную память, но плохо.
- Новая тема: *spike-timing-dependent plasticity* (STDP), т.е. насколько спайки зависят от временных свойств сигналов.

От перцептрона к нейронной сети

Суть происходящего

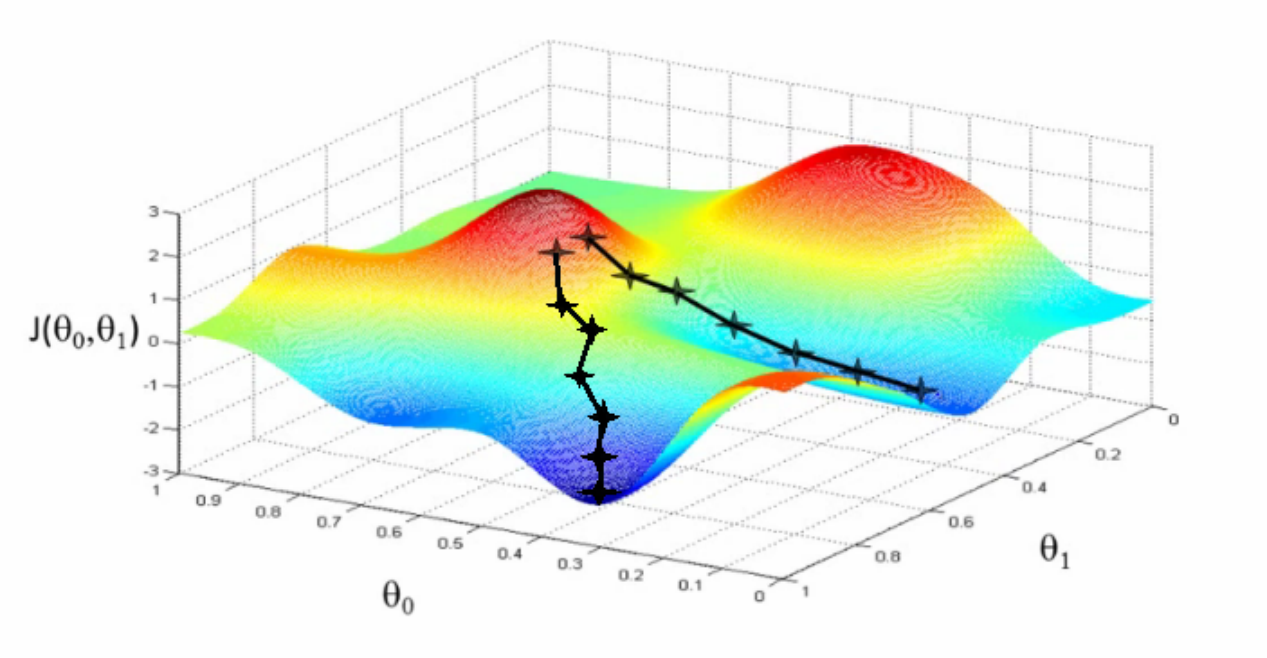
- Вообще говоря, всё, что мы делаем в машинном обучении, — это аппроксимация и оптимизация функций.
- Например:
  - есть функция из картинок в то, что на них изображено;
  - ещё проще: бинарная функция из пикселей картинки в «котик или не котик»;
  - функция, мягко скажем, довольно сложная; задана в виде датасета; как нам её реализовать?
- Обычный подход в машинном обучении:
  - строим какой-то класс моделей, обычно параметрический;
  - и подгоняем параметры так, чтобы модель хорошо описывала данные;
  - то есть оптимизируем какую-то функцию ошибки или функцию качества (правдоподобие, апостериорную вероятность).
- Нейронные сети — это очень мощный и гибкий класс таких моделей.
- Сложность в том, как же обучить их параметры.

Машинное обучение как оптимизация

- Итак, в машинном обучении много разных задач, с учителем и без.
- Их обычно решают по теореме Байеса, пересчитывая наши представления о параметрах в зависимости от полученных данных.
- Для этого обычно надо оптимизировать какую-нибудь сложную функцию (например, апостериорное распределение).
- И для невыпуклых функций это обычно делают тем или иным вариантом *градиентного спуска*.

Градиентный спуск

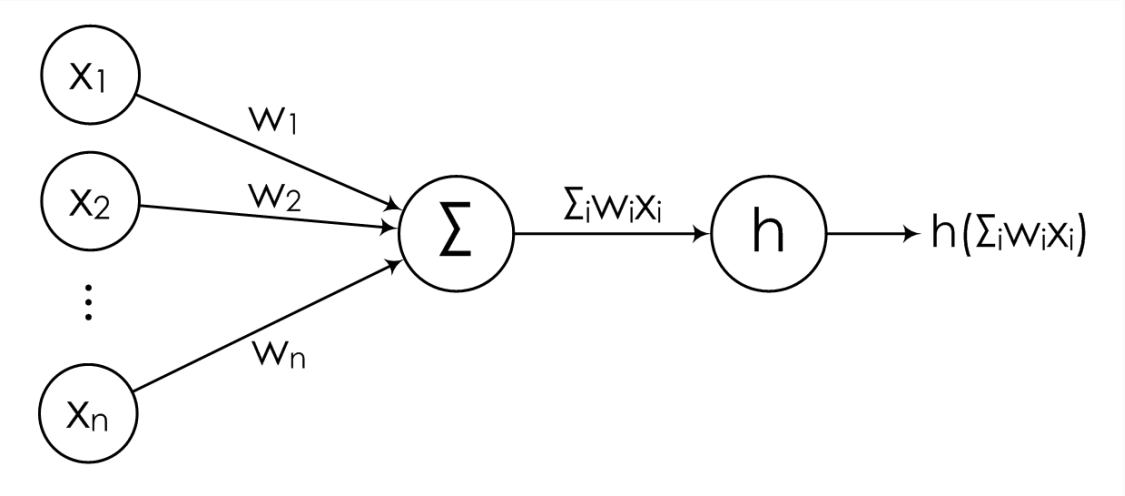
- Градиентный спуск — главный и фактически единственный способ оптимизации очень сложных функций.



- Берём градиент $\nabla E(\mathbf{w})$, сдвигаемся немножко, повторяем.

Перцептрон

- Основной компонент нейронной сети – *перцептрон*:



- Линейная комбинация входов, потом нелинейность:

$$y = h(\mathbf{w}^\top \mathbf{x}) = h\left(\sum_i w_i x_i\right).$$

Перцептрон

- Обучение *градиентным спуском*.
- Для исходного перцептрона Розенблатта ($h = \mathrm{id}$):

$$E_P(\mathbf{w}) = -\sum_{\mathbf{x}\in\mathcal{M}} y(\mathbf{x})\left(\mathbf{w}^\top \mathbf{x}\right),$$

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla_{\mathbf{w}} E_P(\mathbf{w}) = \mathbf{w}^{(\tau)} + \eta t_n \mathbf{x}_n.$$

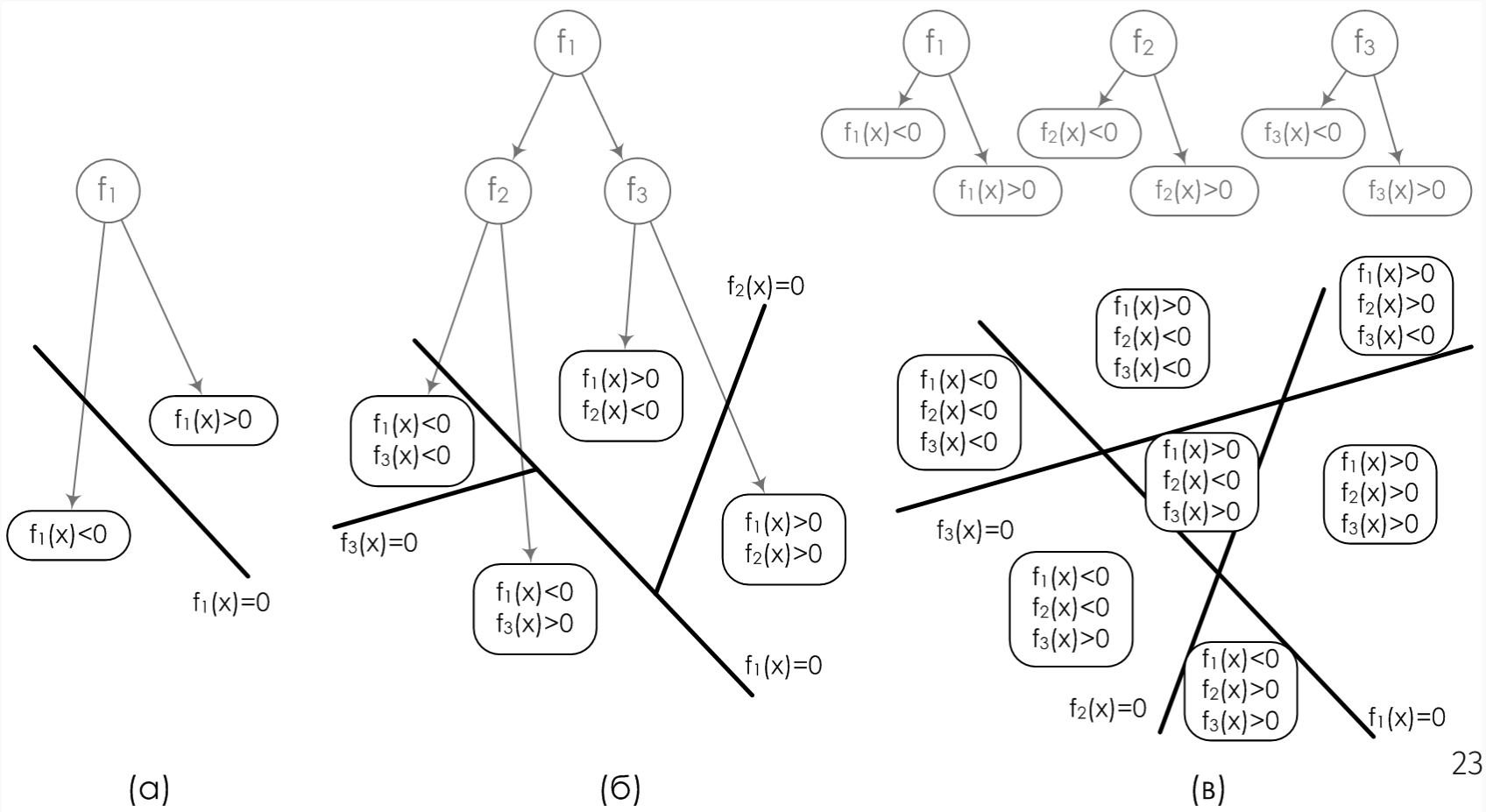
- Или, к примеру, можно делать классификацию, подставляя в качестве функции активации логистический сигмоид $h(x) = \sigma(x) = \frac{1}{1+e^{-x}}$:

$$E(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log\sigma(\mathbf{w}^\top \mathbf{x}_i) + (1-y_i)\log\left(1-\sigma(\mathbf{w}^\top \mathbf{x}_i)\right)\right).$$

- По сути это просто логистическая регрессия.

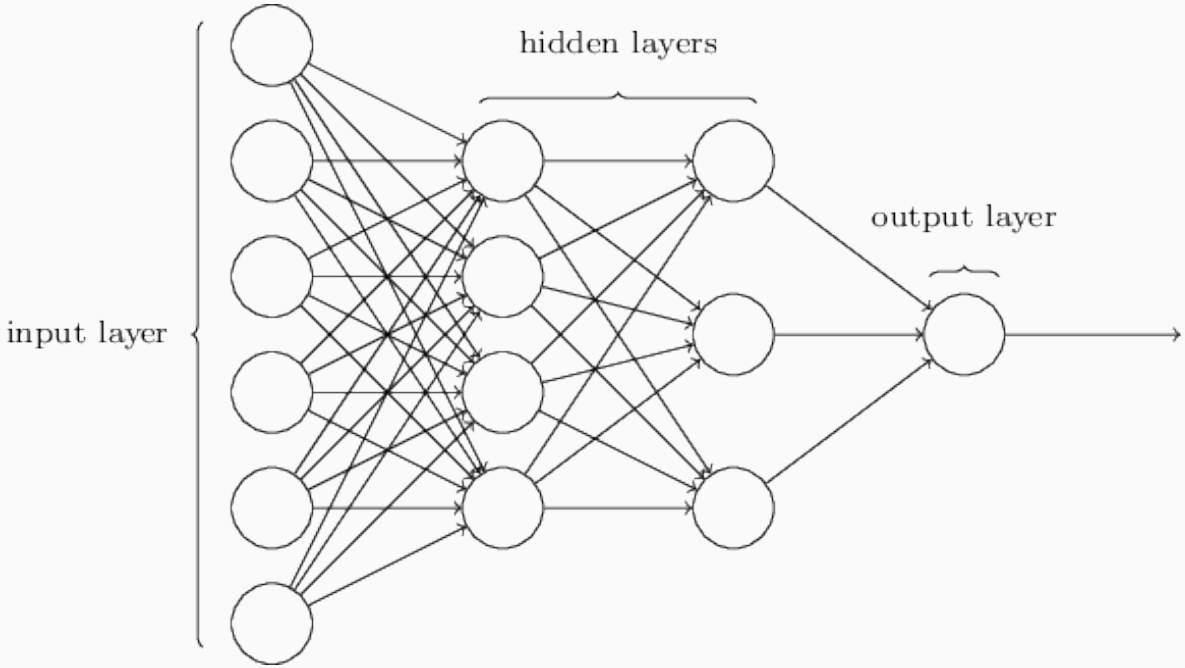
Объединяем перцептроны в сеть

- Сеть перцептронов; выходы одних – входы других.
- Hornik, 1990: двух уровней достаточно для приближения любой функции.
- Но глубокие сети эффективнее — distributed representations:



Объединяем перцептроны в сеть

- Обычно нейронные сети организованы в *слои*.
- Это очень естественно и легко параллелизуется.



Объединяем перцептроны в сеть

- Но по сути мы приближаем очень сложную функцию большой композицией простых функций.
- Как теперь её обучить?
- Ответ прост: градиентный спуск. Но есть и сложности.

Градиентный спуск и графы вычислений

Градиентный спуск

- Градиентный спуск: считаем градиент относительно весов, двигаемся в нужном направлении.
- Формально: для функции ошибки E, целевых значений y и модели f с параметрами θ:

$$E(\theta) = \sum_{(\mathbf{x},y)\in D} E(f(\mathbf{x},\theta),y),$$

$$\theta_t = \theta_{t-1} - \eta\nabla E(\theta_{t-1}) = \theta_{t-1} - \eta \sum_{(\mathbf{x},y)\in D} \nabla E(f(\mathbf{x},\theta_{t-1}),y).$$

- То есть надо по всему датасету пройтись, чтобы хоть куда-то сдвинуться?..

Градиентный спуск

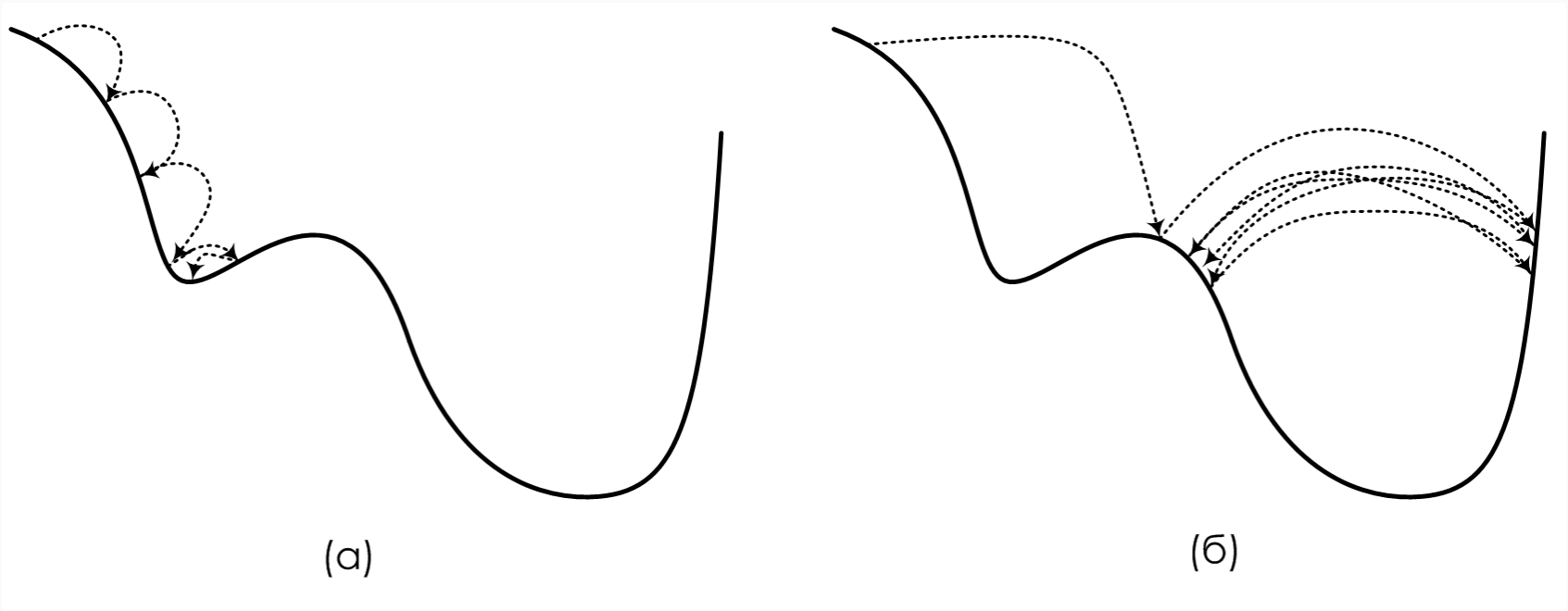
- Нет, конечно — *стохастический градиентный спуск* обновляет после каждого примера:

$$\theta_t = \theta_{t-1} - \eta \nabla E(f(\mathbf{x}_t, \theta_{t-1}), y_t),$$

- А на практике обычно используют *мини-батчи*, их легко параллелизовать и они сглаживают излишнюю “стохастичность”.
- Пока что единственный реальный параметр – это скорость обучения η.

Градиентный спуск

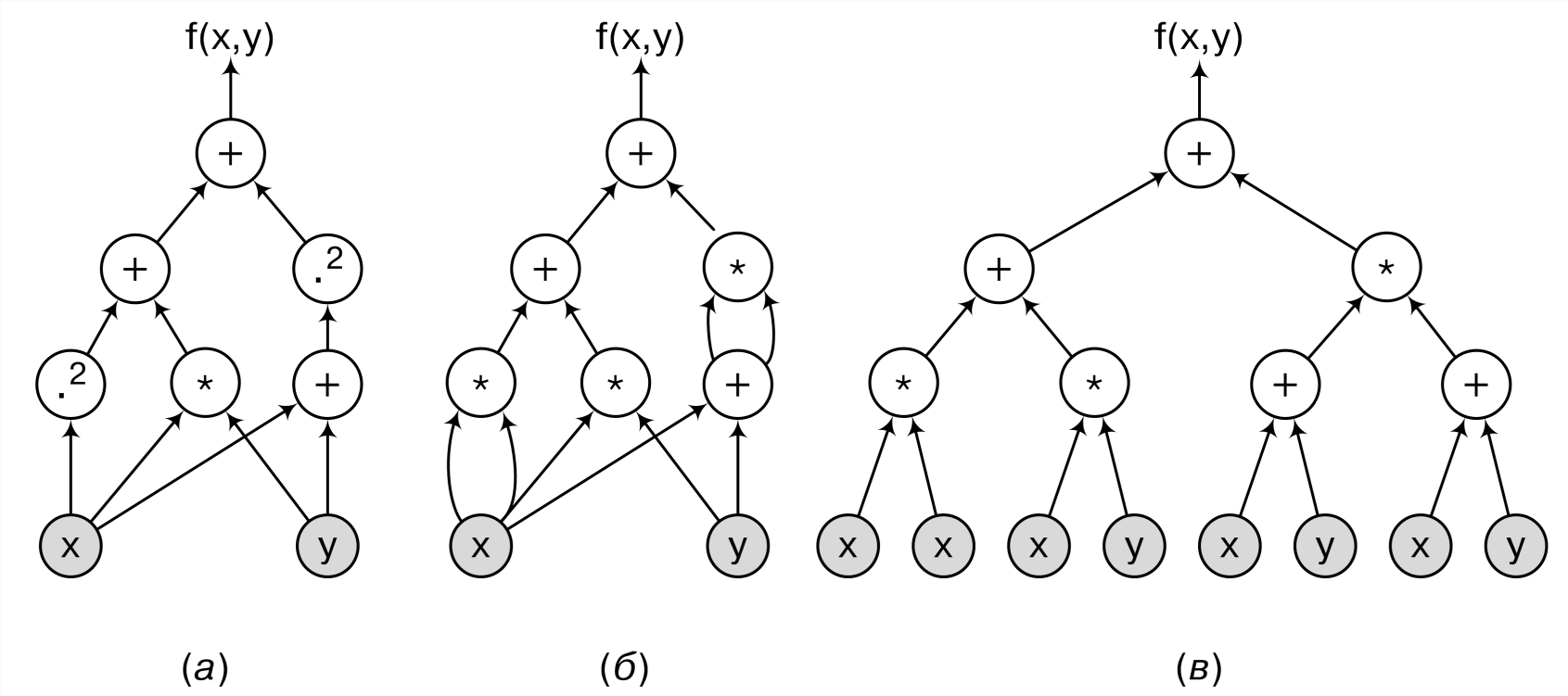
- Со скоростью обучения η масса проблем:



- Мы вернёмся к ним позже, а пока поговорим о производных.

Граф вычислений, fprop and bprop

- Представим функцию как композицию простых функций (т.е. таких, от которых можно производную взять).
- Пример: $f(x, y) = x^2 + xy + (x + y)^2$:



Граф вычислений, frop and bprop

- Градиент теперь можно взять по правилу дифференцирования сложной функции (chain rule):

$$(f \circ g)'(x) = (f(g(x)))' = f'(g(x))g'(x).$$

- По сути это значит, что небольшое изменение δx приводит к

$$\delta f = f'(g(x))\delta g = f'(g(x))g'(x)\delta x.$$

- Нам нужно только уметь брать *градиенты*, т.е. производные по векторам:

$$\nabla_{\mathbf{x}} f = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}.$$

$$\nabla_{\mathbf{x}}(f \circ g) = \begin{pmatrix} \frac{\partial f \circ g}{\partial x_1} \\ \vdots \\ \frac{\partial f \circ g}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \frac{\partial f}{\partial g}\frac{\partial g}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial g}\frac{\partial g}{\partial x_n} \end{pmatrix} = \frac{\partial f}{\partial g}\nabla_{\mathbf{x}} g.$$

Граф вычислений, frop and bprop

- А если f зависит от x несколько раз, $f = f(g_1(x), g_2(x), \dots, g_k(x))$, δx тоже несколько раз появляется:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g_1}\frac{\partial g_1}{\partial x} + \dots + \frac{\partial f}{\partial g_k}\frac{\partial g_k}{\partial x} = \sum_{i=1}^{k}\frac{\partial f}{\partial g_i}\frac{\partial g_i}{\partial x}.$$

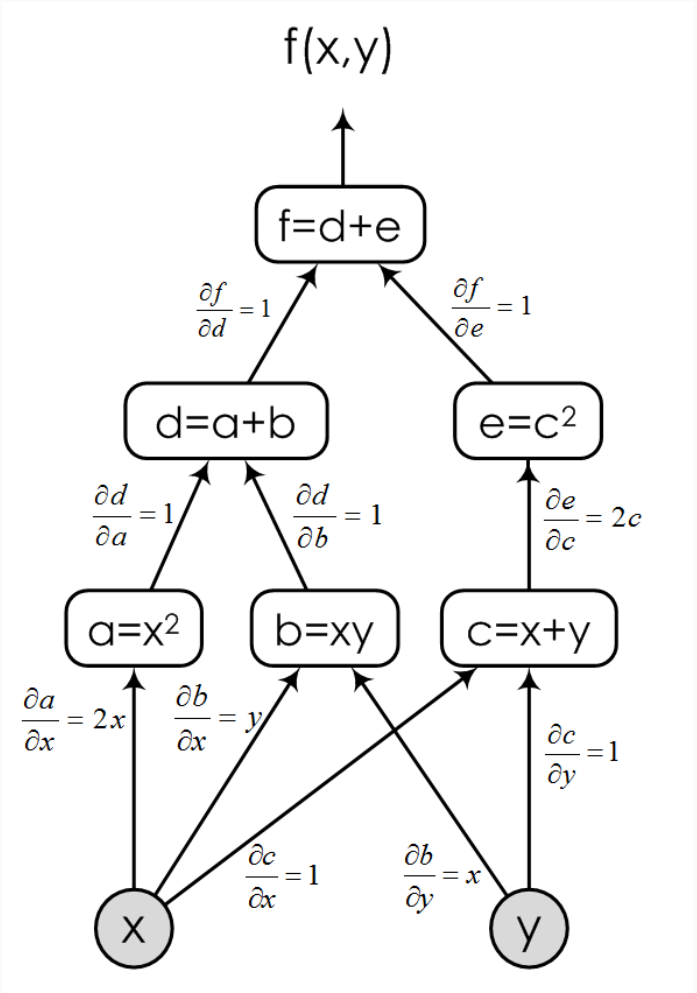
$$\nabla_{\mathbf{x}} f = \frac{\partial f}{\partial g_1}\nabla_{\mathbf{x}} g_1 + \dots + \frac{\partial f}{\partial g_k}\nabla_{\mathbf{x}} g_k = \sum_{i=1}^{k}\frac{\partial f}{\partial g_i}\nabla_{\mathbf{x}} g_i.$$

- Это матричное умножение на *матрицу Якоби*:

$$\nabla_{\mathbf{x}} f = \nabla_{\mathbf{x}}\mathbf{g}\nabla_{\mathbf{g}} f, \text{ где } \nabla_{\mathbf{x}}\mathbf{g} = \begin{pmatrix} \frac{\partial g_1}{\partial x_1} & \dots & \frac{\partial g_k}{\partial x_1} \\ \vdots & & \vdots \\ \frac{\partial g_1}{\partial x_n} & \dots & \frac{\partial g_k}{\partial x_n} \end{pmatrix}.$$

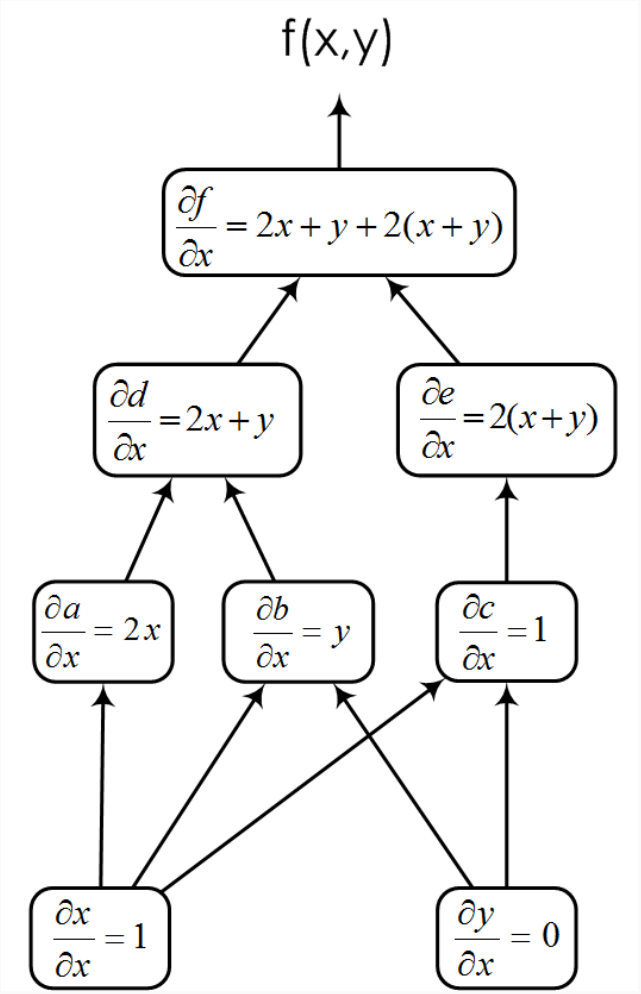
Граф вычислений, fprop and bprop

- Возвращаемся к примеру:



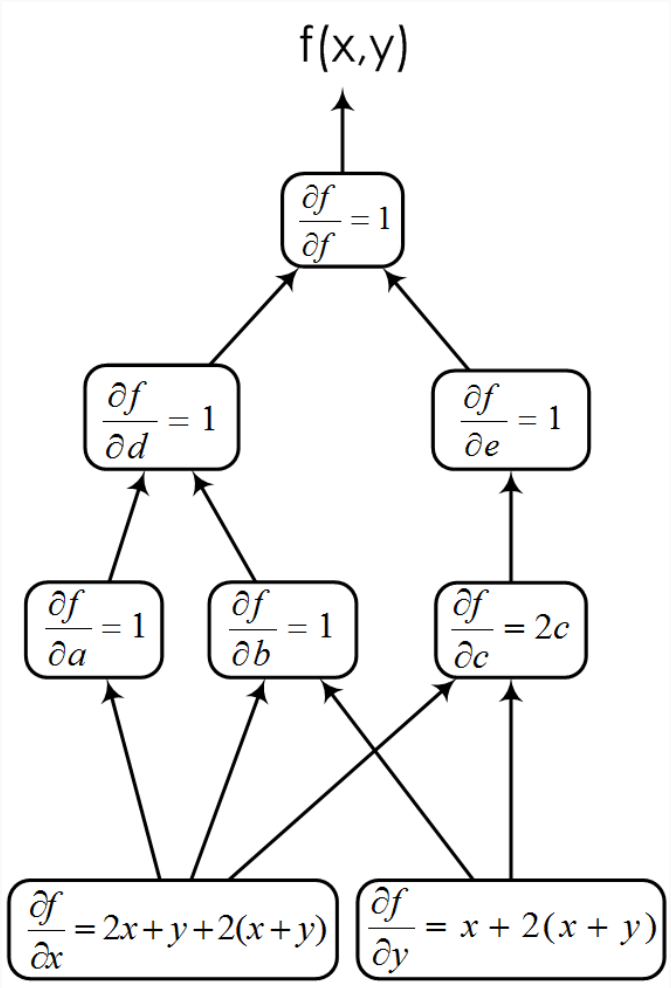
Граф вычислений, frop and bprop

- Прямое распространение (forward propagation, fprop): вычисляем $\frac{\partial f}{\partial x}$ как сложную функцию.



Граф вычислений, fprop and bprop

- Обратное распространение (backpropagation, backprop): начинаем от конца и идём как $\frac{\partial f}{\partial g} = \sum_{g' \in \text{Children}(g)} \frac{\partial f}{\partial g'} \frac{\partial g'}{\partial g}$.

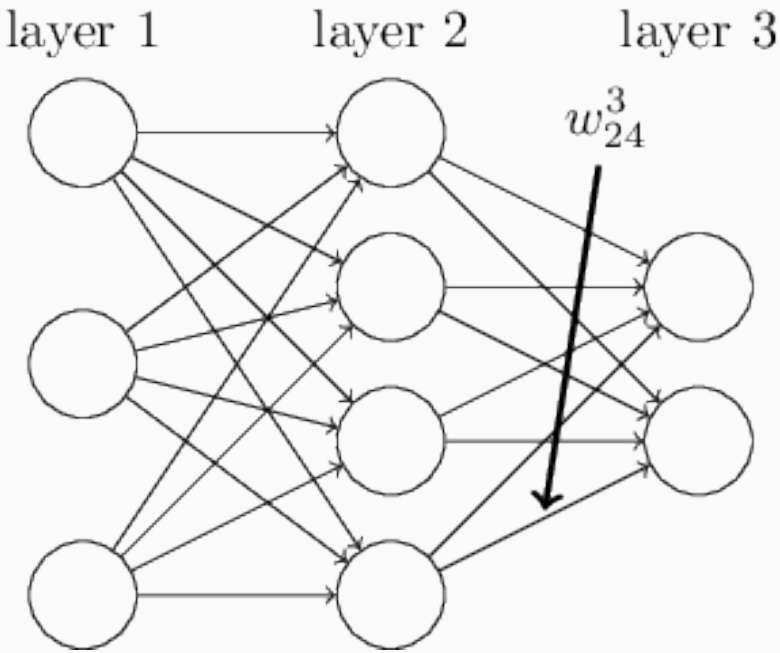


Граф вычислений, fprop and bprop

- Backprop гораздо лучше: получаем все производные за один проход по графу.
- Вот и всё! Теперь мы можем считать градиенты от любых, сколь угодно сложных функций; нужно только, чтобы они представлялись как композиции простых.
- А это всё, что нужно для градиентного спуска!!
- Библиотеки *pyTorch*, *TensorFlow*, *theano* — это на самом деле библиотеки для *автоматического дифференцирования*, это их основная функция.
- И теперь мы можем реализовать массу “классических” моделей в *pyTorch* и обучить их градиентным спуском.
- А у живых нейронов, кстати, не получается, потому что нужно два разных “алгоритма” для вычисления самой функции и градиента.

Граф вычислений, fprop and bprop

- Если сеть организована в слои, то fprop и backprop можно векторизовать, т.е. представить в виде матричных операций.



w^l_{jk} is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer

- Обозначим $w^{(l)}_{jk}$ вес связи от k-го нейрона слоя $(l-1)$ к j-му нейрону слоя l.

Граф вычислений, frop and bprop

- Обозначим $w_{jk}^{(l)}$ вес связи от k-го нейрона слоя $(l-1)$ к j-му нейрону слоя l.
- Также $b_j^{(l)}$ – bias j-го нейрона, $a_j^{(l)}$ – его активация, т.е.

$$a_j^{(l)} = h\left(\sum_k w_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)}\right).$$

- Это можно записать в векторной форме:

$$\mathbf{a}^{(l)} = h\left((\mathbf{w}^{(l)})^\top \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}\right) = h\left(\mathbf{z}^{(l)}\right).$$

- А наша цель – посчитать производные функции ошибки по всем переменным: $\frac{\partial C}{\partial w_{jk}^{(l)}}$, $\frac{\partial C}{\partial b_j^{(l)}}$.

Граф вычислений, frop and bprop

- Определяем ошибку j-го нейрона в слое l:

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}}.$$

- И backprop теперь можно делать от последнего уровня L ко входу, сразу в векторном виде:

$$\delta_j^{(L)} = \frac{\partial C}{\partial a^{(L)}} h'\left(z_j^{(L)}\right), \text{ т.е. } \delta^{(L)} = \nabla_{\mathbf{a}^{(L)}} C \odot h'\left(\mathbf{z}^{(L)}\right),$$

$$\delta^{(l)} = \left((W^{(l+1)})^\top \delta^{(l+1)}\right) \odot h'\left(\mathbf{z}^{(l)}\right),$$

$$\frac{\partial C}{\partial b_j^{(l)}} = \delta_j^{(l)},$$

$$\frac{\partial C}{\partial w_{jk}^{(l)}} = a_k^{(l-1)} \delta_j^{(l)}.$$

- Это всё операции, которые легко параллелизовать на GPU.

Граф вычислений, frop and bprop

- Backpropagation – идея со сложной судьбой.
- Конечно, это просто расчёт градиентов для градиентного спуска.
- Так что уже в 1960-х было понятно, как тащить производные динамическим программированием (и тем более удивительно насчёт Minsky, Papert).
- В явном виде полностью BP для нейронных сетей – M.Sc. thesis (Linnainmaa, 1970).
- (Hinton, 1974) – переоткрыл backprop, популяризовал.
- Rumelhart, Hinton, Williams, “Learning representations by back-propagating errors” (Nature, 1986).

Спасибо за внимание!