ГРАДИЕНТНЫЙ СПУСК В НЕЙРОННЫХ СЕТЯХ

Сергей Николенко

СПбГУ — Санкт-Петербург

13 сентября 2021 г.

Random facts:

- 13 сентября — День программиста в 2021 и других невисокосных годах; он отмечается в 256-й день года, поэтому в високосные годы выпадает на 12 сентября
- 13 сентября в Бразилии — начало недели памяти Революции Фарропилья, республиканского восстания в 1835—1845 гг. в провинциях Риу-Гранди-ду-Сул и Санта-Катарина, а также, что очень удобно, национальный день бразильского рома кашаса
- 13 сентября 1741 г. в Российской империи рабочий день на фабриках был ограничен 15 часами, а 13 сентября 1965 г. в СССР начали бесплатно выдавать молоко сотрудникам вредных производств
- 13 сентября 1979 г. ЮАР провозгласило независимость Республики Венда, бантустана в провинции Трансвааль (сейчас Лимпопо); независимость Венда не была признана ООН и другими странами, и в 1994 г. она влилась обратно в ЮАР

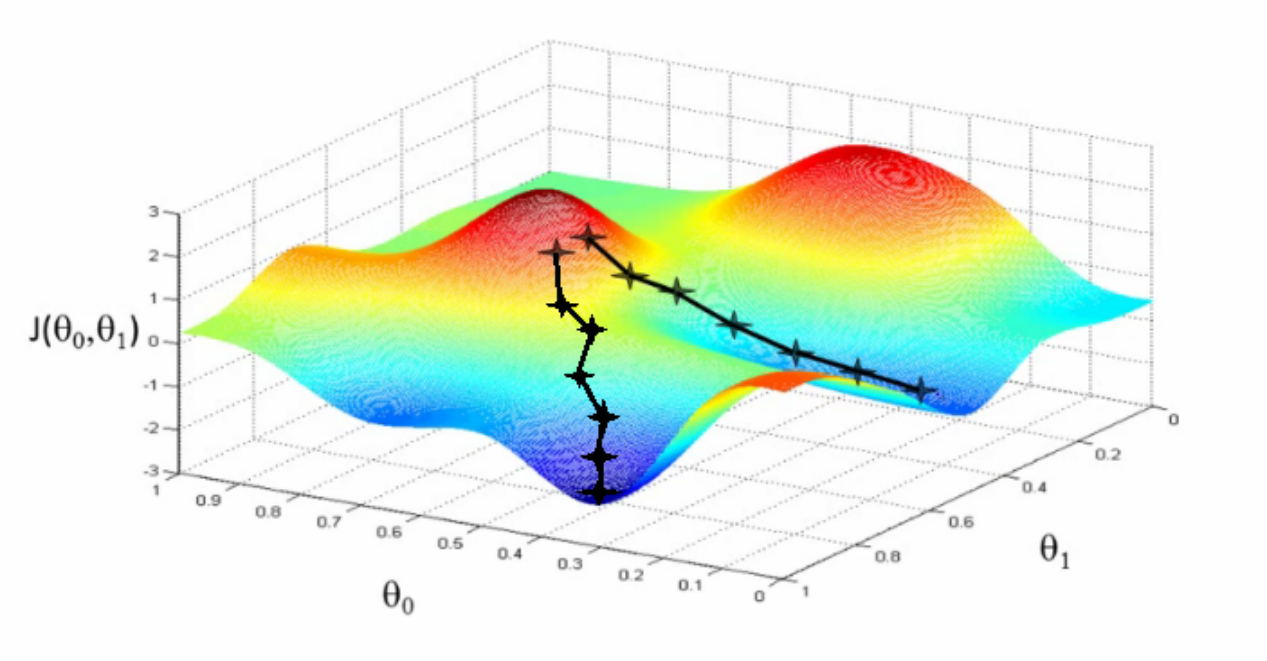
Градиентный спуск и графы вычислений

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

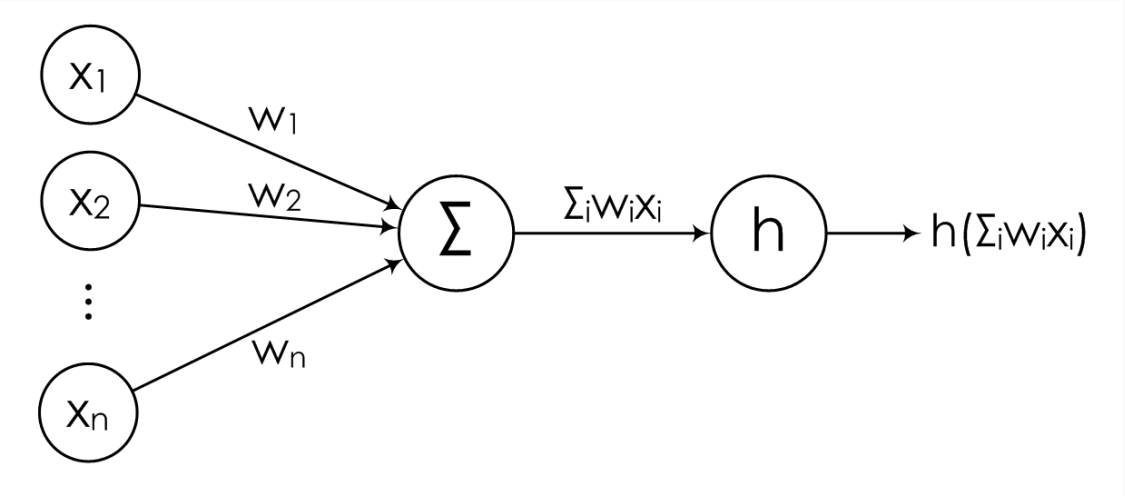
- Градиентный спуск — главный и фактически единственный способ оптимизации очень сложных функций.



- 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 ($h = \mathrm{id}$):

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

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

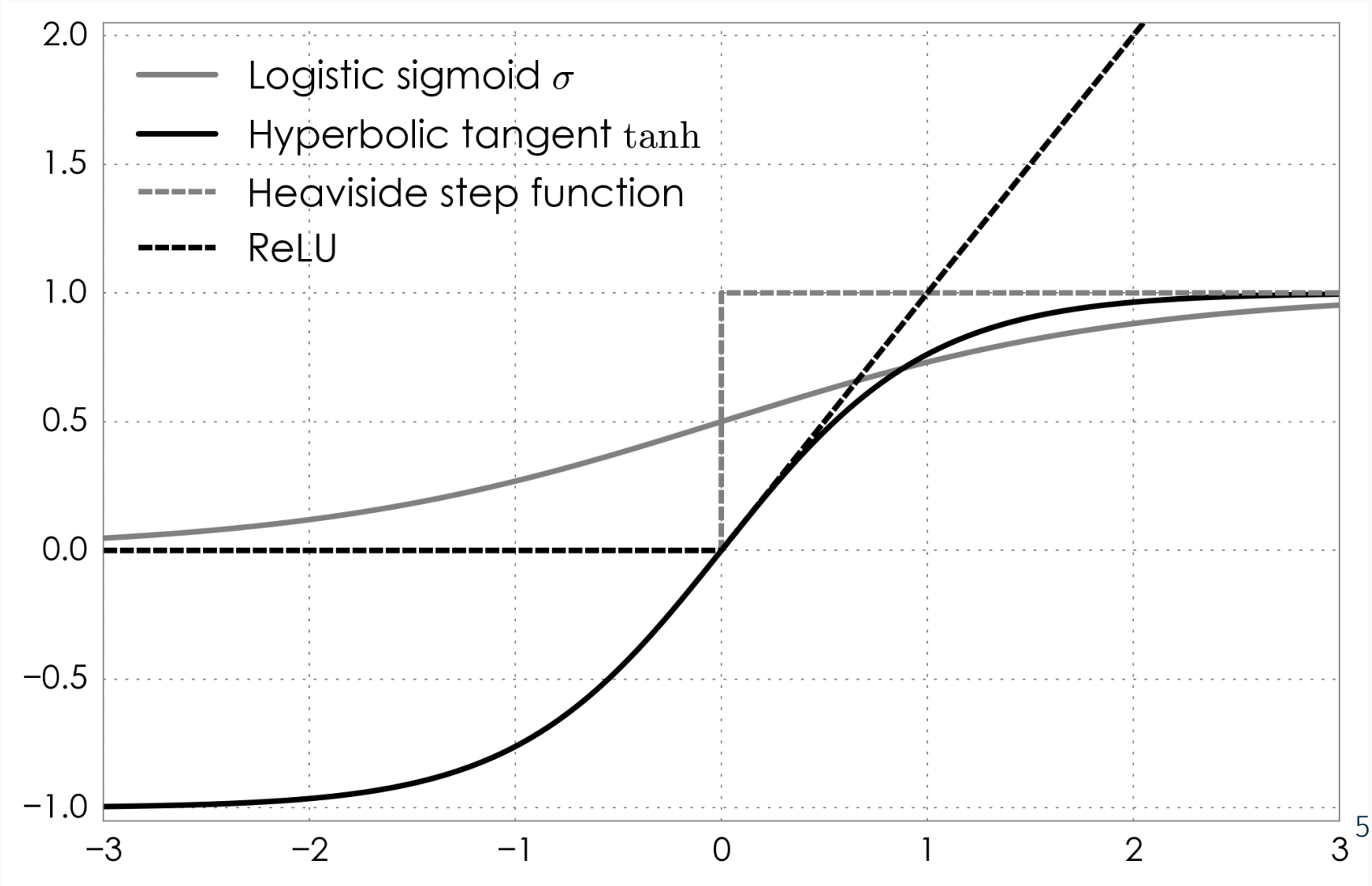
- Или, к примеру, можно делать классификацию, подставляя в качестве функции активации логистический сигмоид $h(x) = \sigma(x) = \frac{1}{1+e^{-x}}$:

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

- По сути это просто логистическая регрессия.

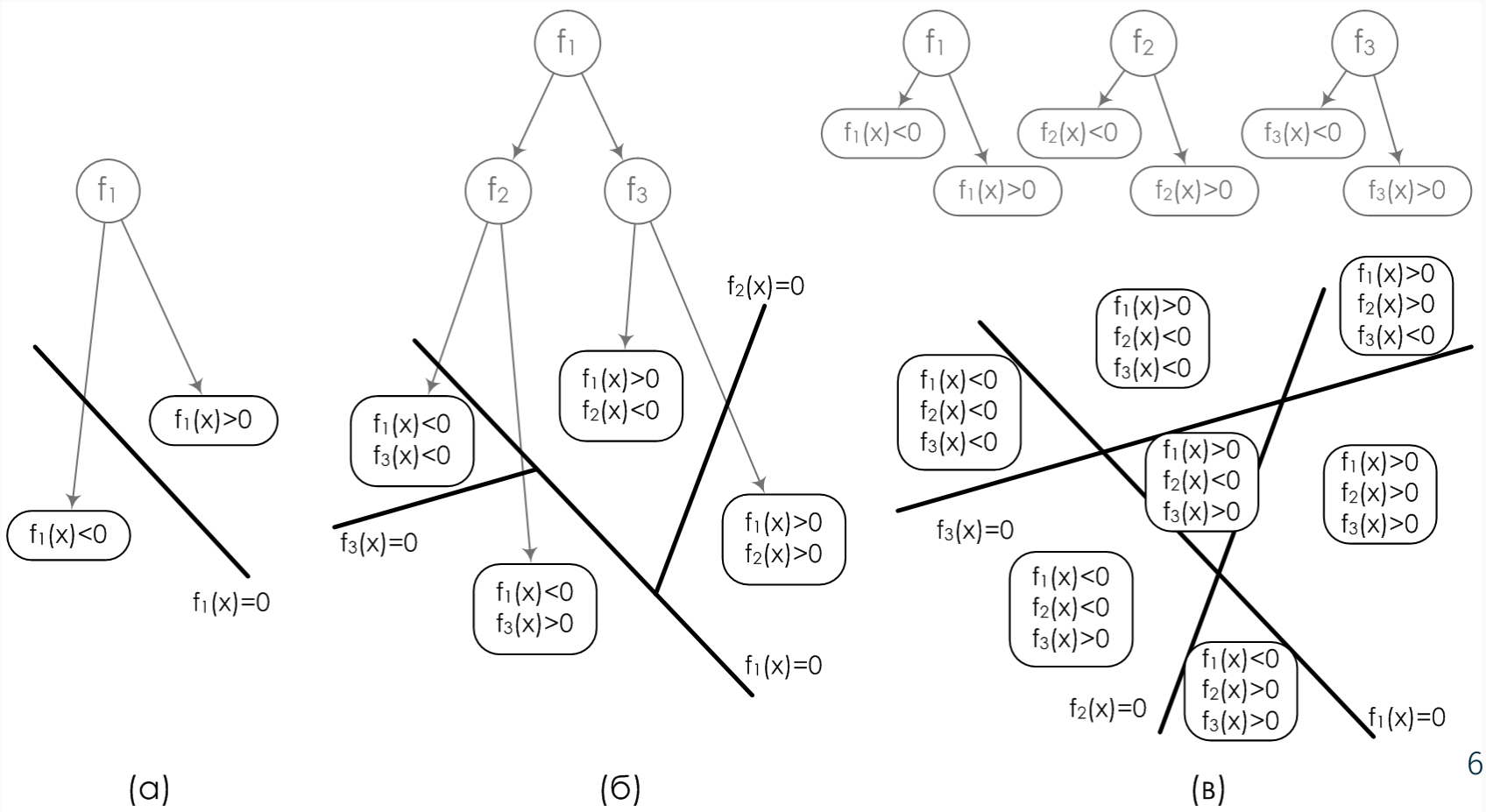
Перцептрон

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



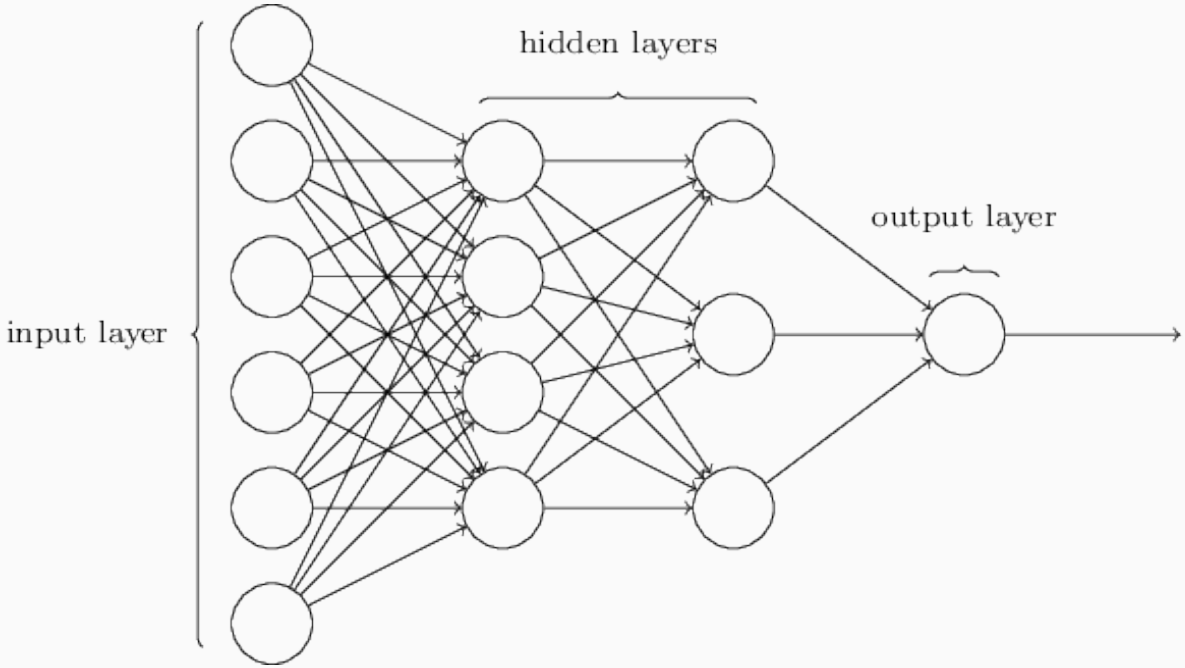
Объединяем перцептроны в сеть

- 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

- 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

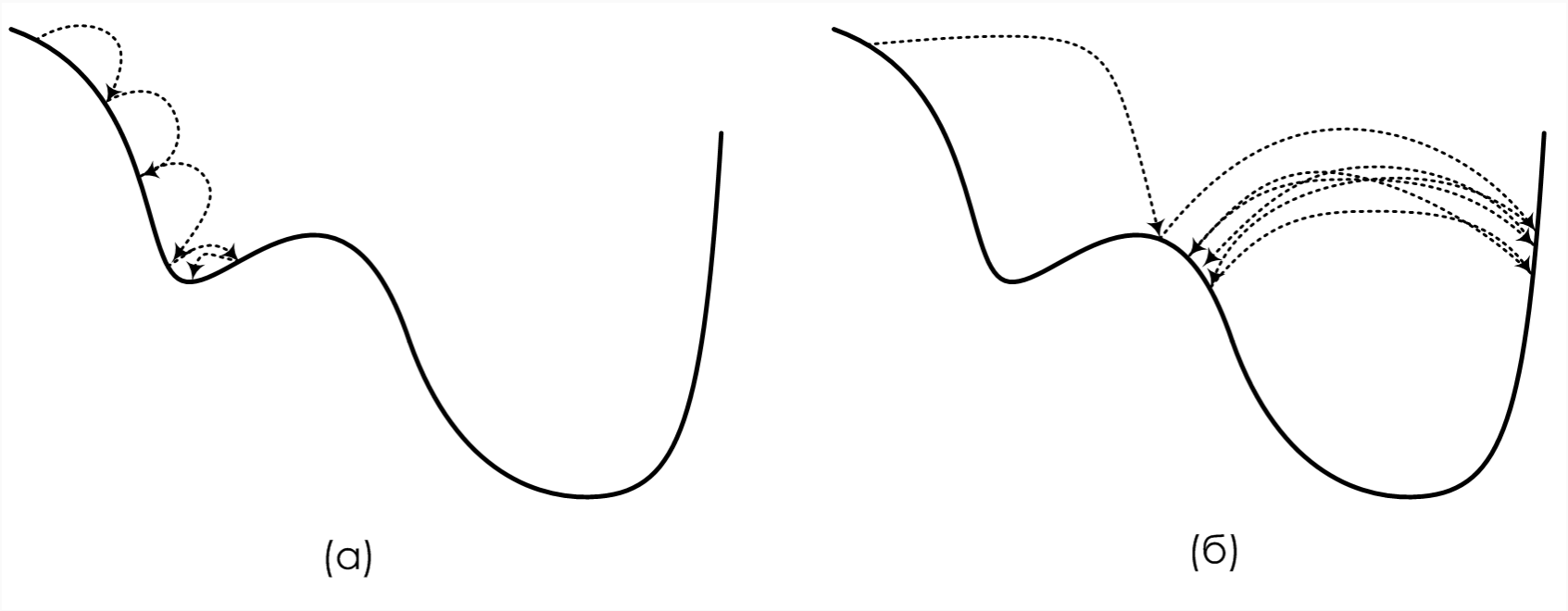
- Нет, конечно — *стохастический градиентный спуск* обновляет после каждого примера:

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

- 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

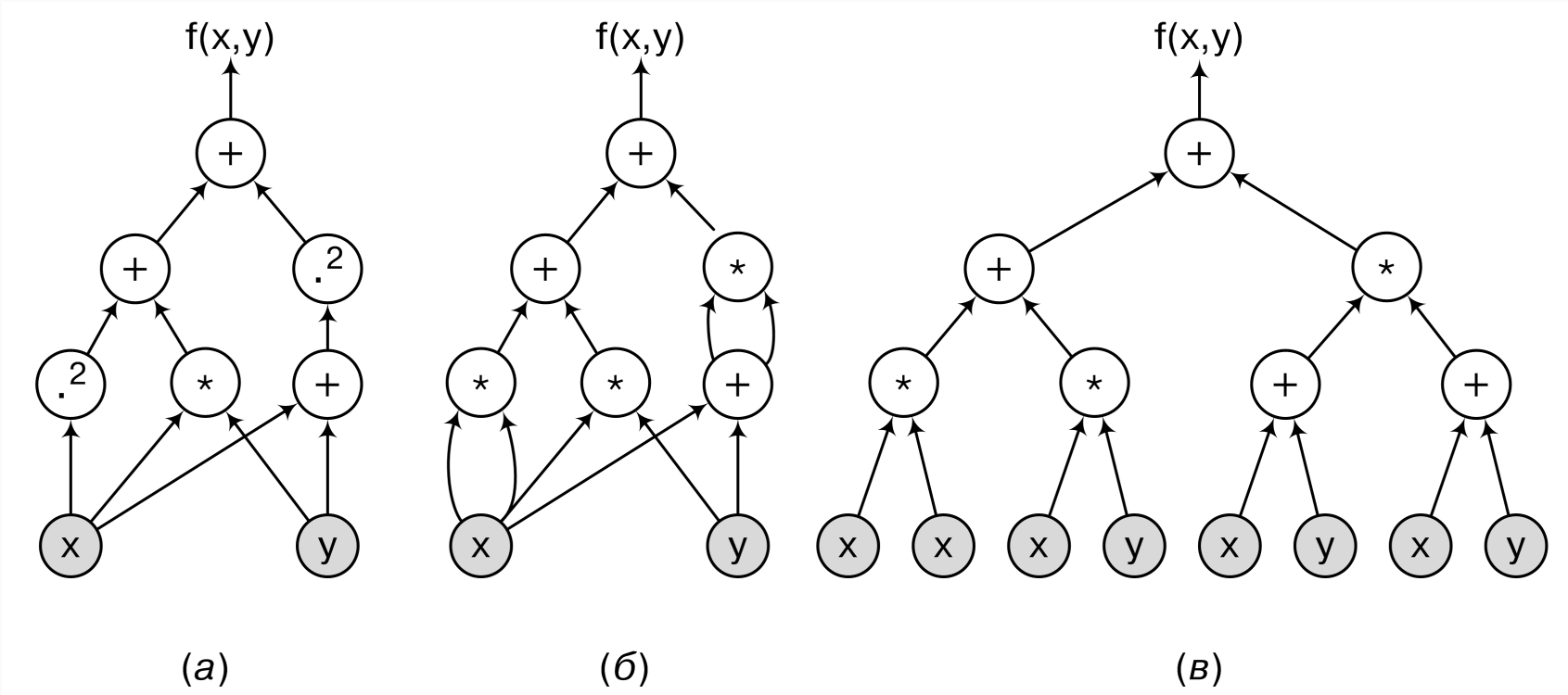
- Со скоростью обучения η масса проблем:



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

Граф вычислений, frop 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, fprop 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:

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

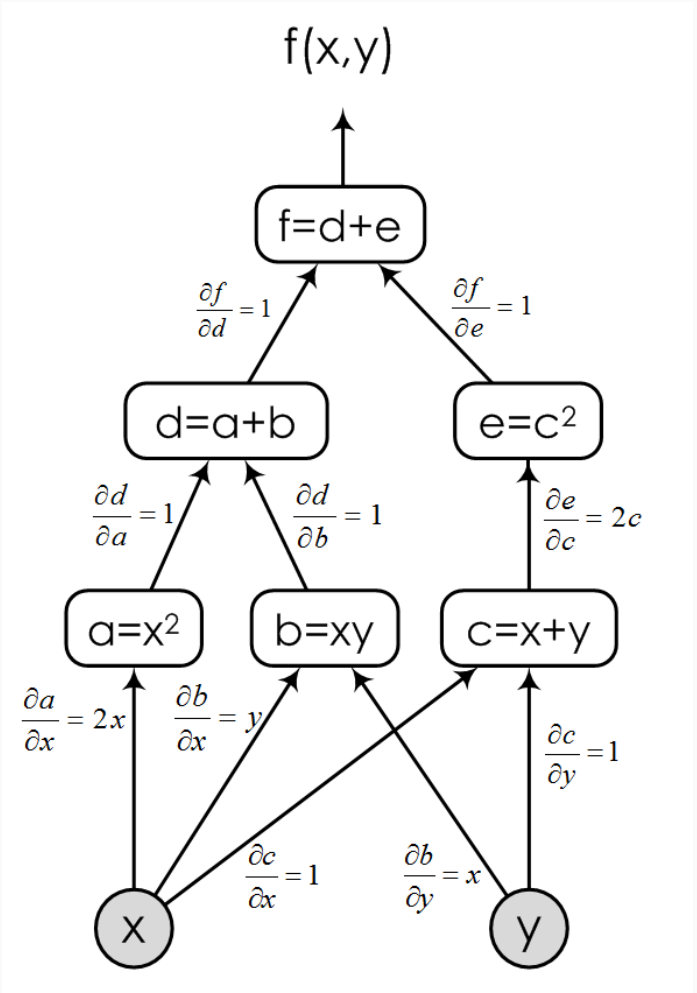
$$\nabla_{\mathbf{x}} f = \frac{\partial f}{\partial g_1}\nabla_{\mathbf{x}} g_1 + \dots + \frac{\partial f}{\partial g_k}\nabla_{\mathbf{x}} g_k = \sum_{i=1}^{k}\frac{\partial f}{\partial g_i}\nabla_{\mathbf{x}} g_i.$$

- Это матричное умножение на *матрицу Якоби*:

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

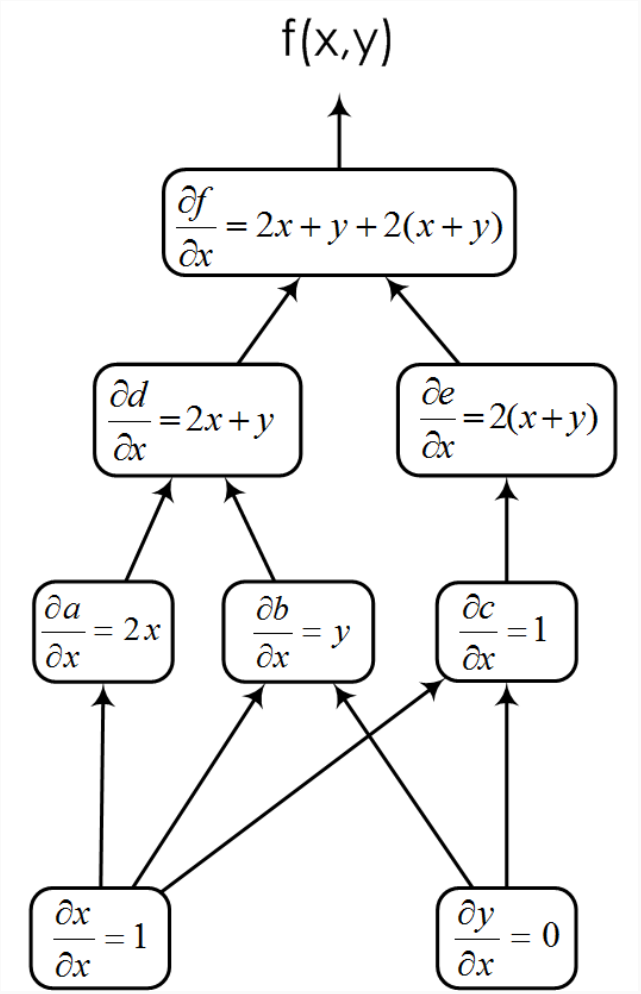
Граф вычислений, frop and bprop

- Возвращаемся к примеру:



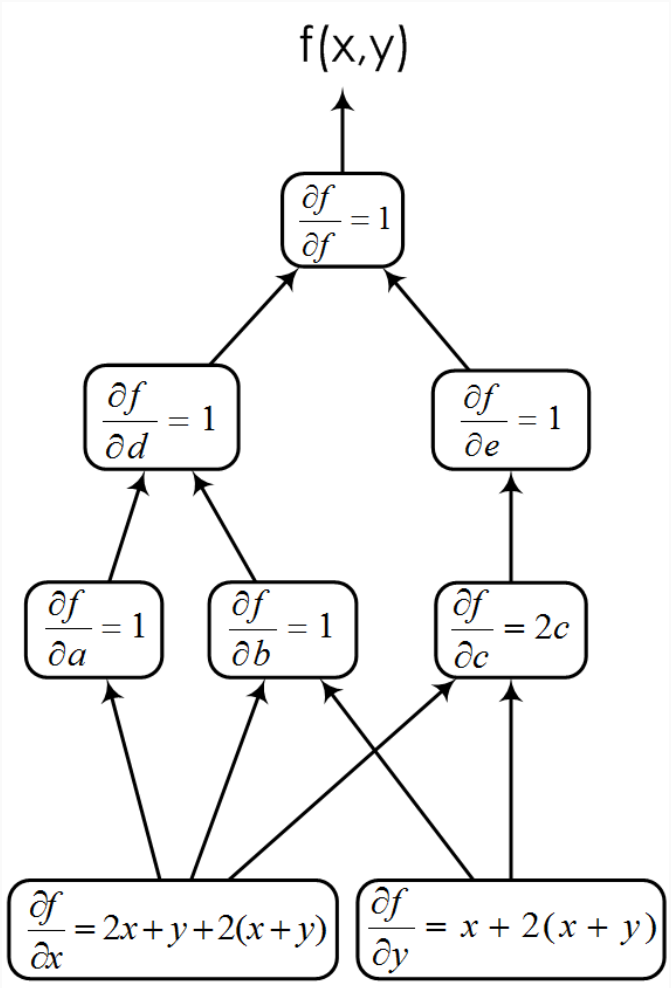
Граф вычислений, frop and bprop

- 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 $\frac{\partial f}{\partial g} = \sum_{g' \in \text{Children}(g)} \frac{\partial f}{\partial g'} \frac{\partial g'}{\partial g}$.

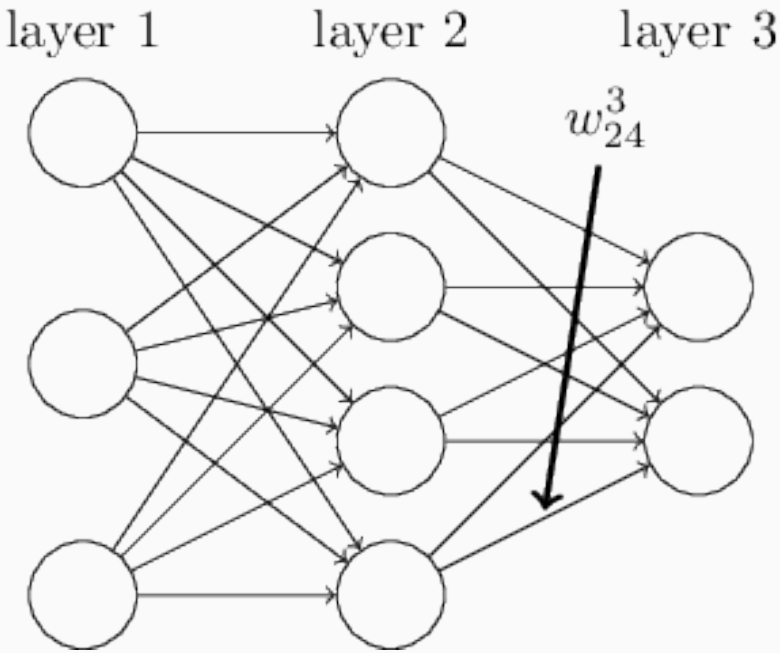


Граф вычислений, frop and bprop

- 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, fprop 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).

Варианты градиентного спуска

Градиентный спуск

- «Ванильный» градиентный спуск:

$$\mathbf{x}_k = \mathbf{x}_{k-1} - \alpha \nabla f(\mathbf{x}_k).$$

- Всё зависит от скорости обучения α.
- Первая мысль — пусть α уменьшается со временем:
  - линейно (linear decay):

$$\alpha = \alpha_0 \left(1 - \frac{t}{T}\right);$$

  - или экспоненциально (exponential decay):

$$\alpha = \alpha_0 e^{-\frac{t}{T}}.$$

Градиентный спуск

- Об этом есть большая наука. Например, условия Вольфе (Wolfe conditions): если мы решаем задачу минимизации $\min_{\mathbf{x}} f(\mathbf{x})$, и на шаге k уже нашли направление $\mathbf{p}_k$, в котором двигаться (например, $\mathbf{p}_k = \nabla_{\mathbf{x}} f(\mathbf{x}_k)$), т.е. надо решить $\min_\alpha f(\mathbf{x}_k + \alpha \mathbf{p}_k)$, то:
  - для $\phi_k(\alpha) = f(\mathbf{x}_k + \alpha \mathbf{p}_k)$ будет $\phi_k'(\alpha) = \nabla f(\mathbf{x}_k + \alpha \mathbf{p}_k)^\top \mathbf{p}_k$, и если $\mathbf{p}_k$ – направление спуска, то $\phi_k'(0) < 0$;
  - шаг α должен удовлетворять условиям Армихо (Armijo rule):

$$\phi_k(\alpha) \le \phi_k(0) + c_1 \alpha \phi_k'(0) \text{ для некоторого } c_1 \in (0, \frac{1}{2});$$

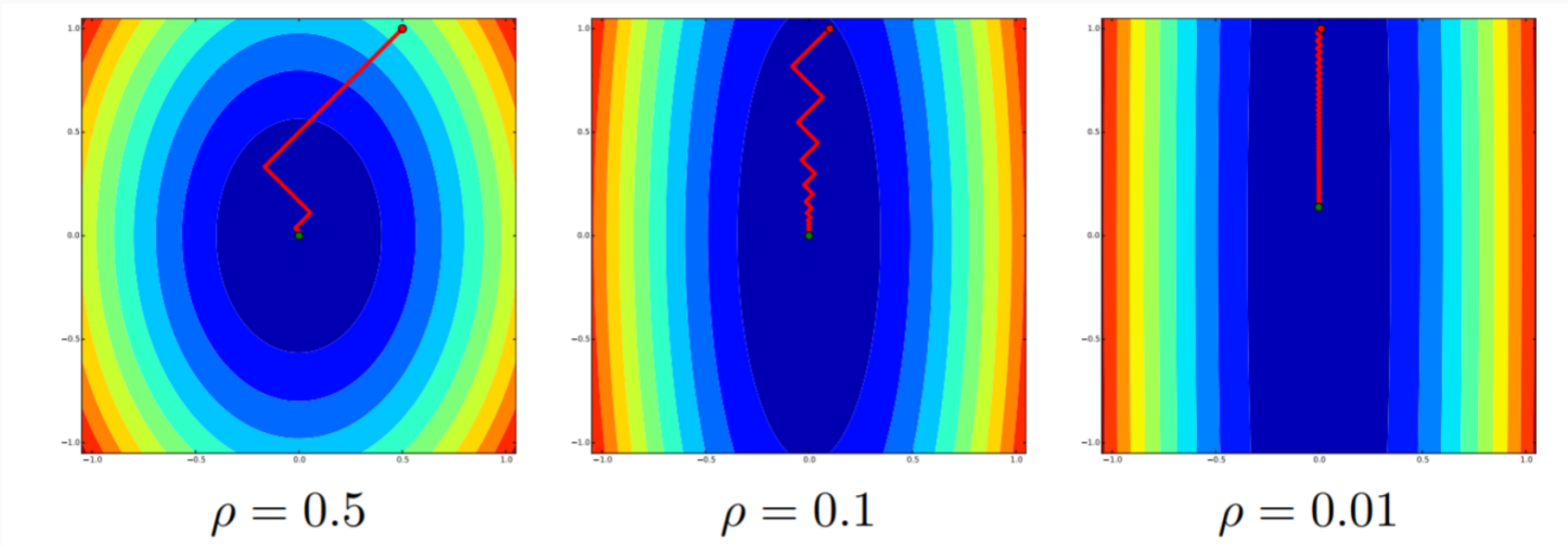
  - или даже более сильным условиям Вульфа (Wolfe rule): Армихо плюс

$$|\phi_k'(\alpha)| \le c_2 |\phi_k'(0)|,$$

    т.е. мы хотим уменьшить проекцию градиента.
- Останавливаем когда $\|\nabla_{\mathbf{x}} f(\mathbf{x}_k)\|^2 \le \epsilon$ или $\|\nabla_{\mathbf{x}} f(\mathbf{x}_k)\|^2 \le \epsilon \|\nabla_{\mathbf{x}} f(\mathbf{x}_0)\|^2$ (а почему квадрат, кстати?).

Градиентный спуск

- Давайте посмотрим, что происходит, если масштаб разный: для функции $f(x, y) = \frac{1}{2}x^2 + \frac{\rho}{2}y^2 \to \min_{x,y}$



$\rho = 0.5$ $\rho = 0.1$ $\rho = 0.01$

- Для вытянутых «долин» (переменных с разным масштабированием) мы сразу получаем кучу лишних итераций, очень медленно.
- Лучше быть *адаптивным*; как это сделать?

Градиентный спуск

- Лучше всего, конечно, *метод Ньютона*: давайте отмасштабируем обратно при помощи гессиана

$$\mathbf{g}_k = \nabla_{\mathbf{x}} f(\mathbf{x}_k),\ H_k = \nabla^2_{\mathbf{x}} f(\mathbf{x}_k),\ \text{и } \mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k H_k^{-1}\mathbf{g}_k.$$

- Здесь тоже применимо условие Армихо:

$$\alpha_k: \quad f(\mathbf{x}_{k+1}) \le f(\mathbf{x}_k) - c_1\alpha_k g_k^\top H_k^{-1}\mathbf{g}_k,\ c_1 \approx 10^{-4}.$$

- Было бы круто! Но H_k посчитать просто нереально.

Градиентный спуск

- Есть, правда, приближения.
- Метод сопряжённых градиентов, квази-ньютоновские методы...
- L-BFGS (limited memory Broyden–Fletcher–Goldfarb–Shanno):
  - строим аппроксимацию к H^{-1};
  - для этого сохраняем последовательно апдейты аргументов функции и градиентов и выражаем через них H^{-1}.
- Интересный открытый вопрос: можно ли заставить L-BFGS работать для deep learning?
- Но пока не получается, в основном потому, что всё-таки нужно уметь считать градиент.
- А ведь у нас обычно нет возможности даже градиент вычислить...

Стохастический градиентный спуск

- У нас обычно стохастический градиентный спуск:

$$\mathbf{x}_t = \mathbf{x}_{t-1} - \alpha \nabla f(\mathbf{x}_t, \mathbf{x}_{t-1}, y_t).$$

- Да ещё и с мини-батчами; как это понять формально?
- Мы обычно решаем задачу *стохастической оптимизации*:

$$F(\mathbf{x}) = \mathbb{E}_{q(\mathbf{y})} f(\mathbf{x}, \mathbf{y}) \to \min_{\mathbf{x}}:$$

  - минимизация эмпирического риска

$$F(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} f_i(\mathbf{x}) = \mathbb{E}_{i \sim \mathrm{U}(1,\ldots,N)} f_i(\mathbf{x}) \to \min_{\mathbf{x}};$$

  - минимизация вариационной нижней оценки (ELBO)... но об этом позже.

- Что такое теперь, получается, мини-батчи?

Стохастический градиентный спуск

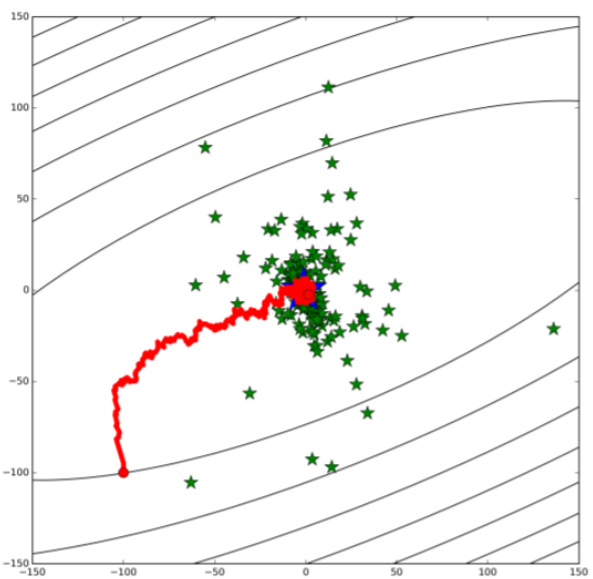
- Да просто эмпирические оценки общей функции по подвыборке:

$$\hat{F}(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m} f(\mathbf{x}, \mathbf{y}_i), \quad \hat{\mathbf{g}}(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m} \nabla_{\mathbf{x}} f(\mathbf{x}, \mathbf{y}_i).$$

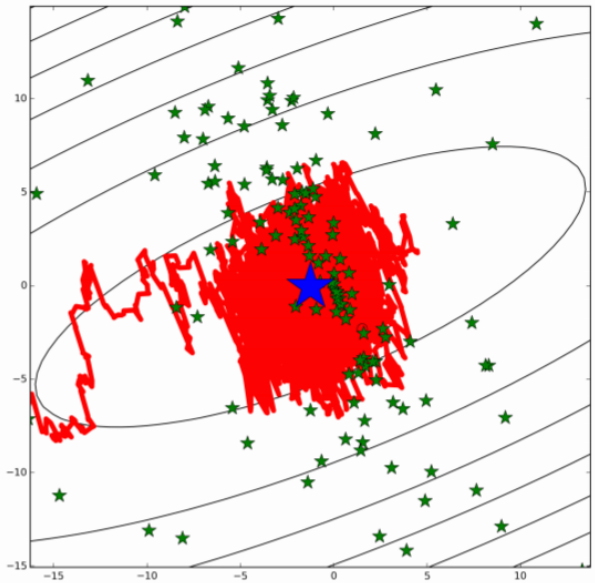
- Это очень хорошие оценки: несмещённые, сходятся на бесконечности (правда, медленно), легко посчитать.
- В целом, так и мотивируется стохастический градиентный спуск (SGD): метод Монте-Карло по сути.
- Но есть проблемы...

Стохастический градиентный спуск

- Проблемы SGD:
  - никогда не идёт в правильном направлении,
  - шаг не равен нулю в оптимуме $F(\mathbf{x})$, т.е. не может сойтись с постоянной длиной шага,
  - мы не знаем ни $F(\mathbf{x})$, ни $\nabla F(\mathbf{x})$, т.е. не можем использовать правила Армихо и Вульфа.



SGD trajectory



optimum vicinity

Стохастический градиентный спуск

- Тем не менее, можно попробовать проанализировать итерацию SGD для $F(\mathbf{x}) = \mathbb{E}_{q(\mathbf{y})} f(\mathbf{x}, \mathbf{y}) \to \min_{\mathbf{x}}$:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \hat{\mathbf{g}}_k, \quad \mathbb{E}\hat{\mathbf{g}}_k = \mathbf{g}_k = \nabla F(\mathbf{x}_k).$$

- Давайте оценим невязку точки на очередной итерации:

$$\begin{aligned}\|\mathbf{x}_{k+1} - \mathbf{x}_{\text{opt}}\|^2 &= \|\mathbf{x}_k - \alpha_k \hat{\mathbf{g}}_k - \mathbf{x}_{\text{opt}}\|^2 = \\ &= \|\mathbf{x}_k - \mathbf{x}_{\text{opt}}\|^2 - 2\alpha_k \hat{\mathbf{g}}_k^\top (\mathbf{x}_k - \mathbf{x}_{\text{opt}}) + \alpha_k^2 \|\hat{\mathbf{g}}_k\|^2.\end{aligned}$$

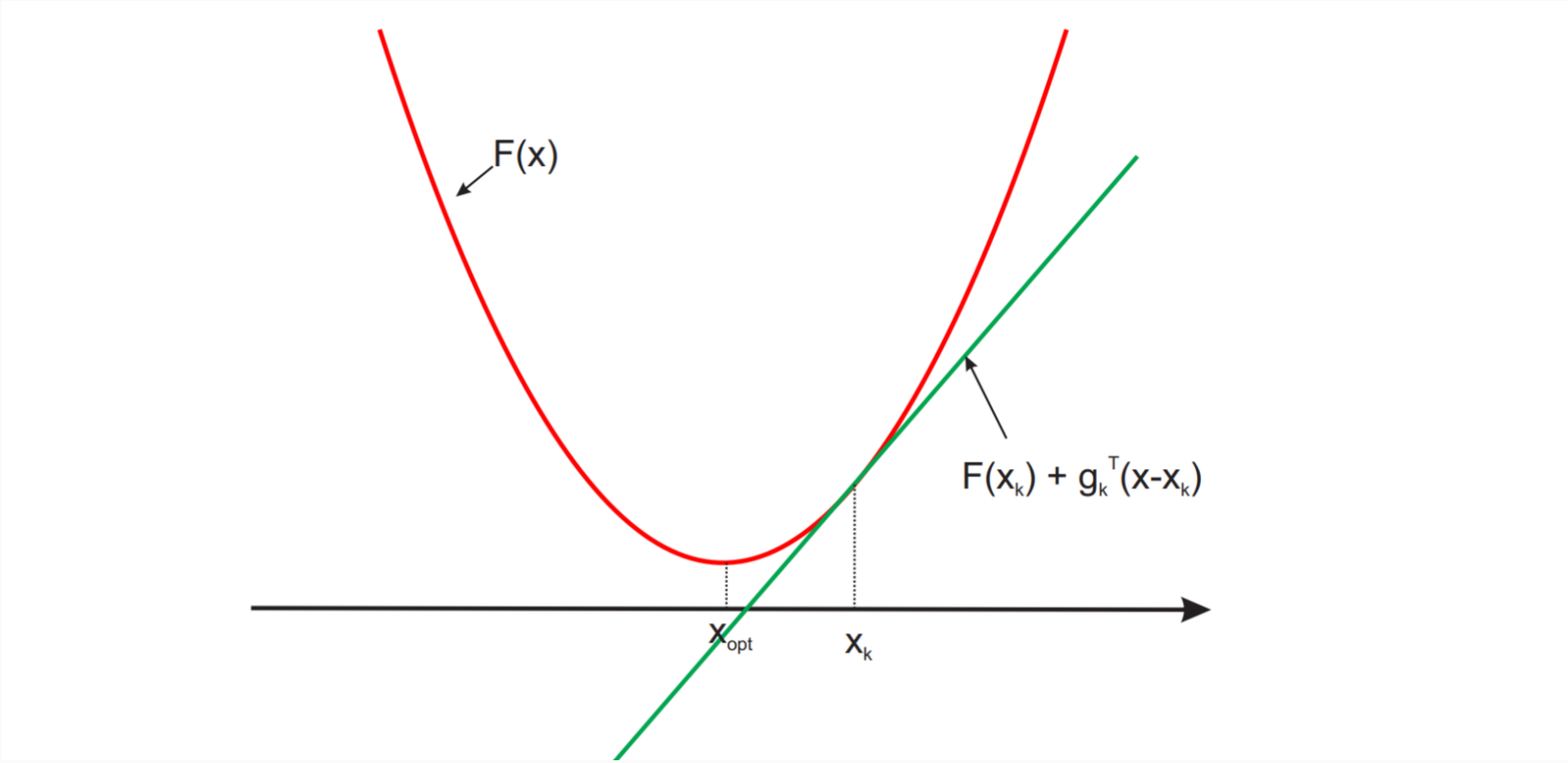
- Возьмём ожидание по $q(\mathbf{y})$ в момент времени k:

$$\mathbb{E}\|\mathbf{x}_{k+1} - \mathbf{x}_{\text{opt}}\|^2 = \|\mathbf{x}_k - \mathbf{x}_{\text{opt}}\|^2 - 2\alpha_k \mathbf{g}_k^\top (\mathbf{x}_k - \mathbf{x}_{\text{opt}}) + \alpha_k^2 \mathbb{E}\|\hat{\mathbf{g}}_k\|^2.$$

Стохастический градиентный спуск

- Для простоты предположим, что F выпуклая:

$$F(\mathbf{x}_{\text{opt}}) \geq F(\mathbf{x}_k) + \mathbf{g}_k^\top (\mathbf{x}_k - \mathbf{x}_{\text{opt}})$$



Стохастический градиентный спуск

- У нас было

$$\mathbb{E}\|\mathbf{x}_{k+1} - \mathbf{x}_{\text{opt}}\|^2 = \|\mathbf{x}_k - \mathbf{x}_{\text{opt}}\|^2 - 2\alpha_k \mathbf{g}_k^\top(\mathbf{x}_k - \mathbf{x}_{\text{opt}}) + \alpha_k^2 \mathbb{E}\|\hat{\mathbf{g}}_k\|^2,$$
$$F(\mathbf{x}_{\text{opt}}) \geq F(\mathbf{x}_k) + \mathbf{g}_k^\top(\mathbf{x}_k - \mathbf{x}_{\text{opt}}).$$

- Значит,

$$\alpha_k(F(\mathbf{x}_k) - F(\mathbf{x}_{\text{opt}})) \leq \alpha_k \mathbf{g}_k^\top(\mathbf{x}_k - \mathbf{x}_{\text{opt}}) =$$
$$= \frac{1}{2}\|\mathbf{x}_k - \mathbf{x}_{\text{opt}}\|^2 + \frac{1}{2}\alpha_k^2 \mathbb{E}\|\hat{\mathbf{g}}_k\|^2 - \frac{1}{2}\mathbb{E}\|\mathbf{x}_{k+1} - \mathbf{x}_{\text{opt}}\|^2.$$

Стохастический градиентный спуск

- Возьмём ожидание от левой части и просуммируем:

$$\sum_{i=0}^{k} \alpha_i (\mathbb{E}F(\mathbf{x}_i) - F(\mathbf{x}_{\text{opt}})) \leq$$

$$\leq \frac{1}{2}\|\mathbf{x}_0 - \mathbf{x}_{\text{opt}}\|^2 + \frac{1}{2}\sum_{i=0}^{k} \alpha_i^2 \mathbb{E}\|\hat{\mathbf{g}}_i\|^2 - \frac{1}{2}\mathbb{E}\|\mathbf{x}_{k+1} - \mathbf{x}_{\text{opt}}\|^2 \leq$$

$$\leq \frac{1}{2}\|\mathbf{x}_0 - \mathbf{x}_{\text{opt}}\|^2 + \frac{1}{2}\sum_{i=0}^{k} \alpha_i^2 \mathbb{E}\|\hat{\mathbf{g}}_i\|^2.$$

- Получилась сумма значений функции в разных точках с весами α_i. Что делать?

Стохастический градиентный спуск

- Воспользуемся выпуклостью:

$$\mathbb{E}F\left(\frac{\sum_i \alpha_i \mathbf{x}_i}{\sum_i \alpha_i}\right) - F(\mathbf{x}_{\text{opt}}) \leq$$

$$\leq \frac{\sum_i \alpha_i (\mathbb{E}F(\mathbf{x}_i) - F(\mathbf{x}_{\text{opt}})}{\sum_i \alpha_i} \leq \frac{\frac{1}{2}\|\mathbf{x}_0 - \mathbf{x}_{\text{opt}}\|^2 + \frac{1}{2}\sum_{i=0}^k \alpha_i^2 \mathbb{E}\|\hat{\mathbf{g}}_i\|^2}{\sum_i \alpha_i}.$$

- Т.е. оценка получилась на значение в линейной комбинации точек (поэтому в статьях часто берут среднее/ожидание или линейную комбинацию, а на практике нет разницы или лучше брать последнюю точку)
- Если $\|\mathbf{x}_0 - \mathbf{x}_{\text{opt}}\| \leq R$ и $\mathbb{E}\|\hat{\mathbf{g}}_k\|^2 \leq G^2$, то

$$\mathbb{E}F(\hat{\mathbf{x}}_k) - F(\mathbf{x}_{\text{opt}}) \leq \frac{R^2 + G^2 \sum_{i=0}^k \alpha_i^2}{2\sum_{i=0}^k \alpha_i}.$$

Стохастический градиентный спуск

- Это самая главная оценка про SGD:

$$\mathbb{E}F(\hat{\mathbf{x}}_k) - F(\mathbf{x}_{\text{opt}}) \leq \frac{R^2 + G^2 \sum_{i=0}^{k} \alpha_i^2}{2 \sum_{i=0}^{k} \alpha_i}.$$

- R – оценка начальной невязки, а G – оценка чего-то вроде дисперсии стохастического градиента.
- Например, для постоянного шага $\alpha_i = h$

$$\mathbb{E}F(\hat{\mathbf{x}}_k) - F(\mathbf{x}_{\text{opt}}) \leq \frac{R^2}{2h(k+1)} + \frac{G^2 h}{2} \to_{k\to\infty} \frac{G^2 h}{2}.$$

Стохастический градиентный спуск

- Итоги про SGD:
  - SGD приходит в «регион неопределённости» радиуса $\frac{1}{2}G^2h$, и этот радиус пропорционален длине шага;
  - чем быстрее идём, тем быстрее придём, но регион неопределённости будет больше, т.е. по идее надо уменьшать со временем скорость обучения;
  - SGD сходится медленно: полный GD для выпуклых функций сходится за $O(1/k)$, а SGD – за $O(1/\sqrt{k})$;
  - но далеко от региона неопределённости у нас скорость тоже $O(1/k)$ получилась для постоянной скорости обучения, т.е. замедляется только уже близко к оптимуму, и вообще цель наша – достичь региона неопределённости;
  - но всё равно всё зависит от G, и это будет особенно важно потом в нейробайесовских методах.

Спасибо!

Спасибо за внимание!