

Общие замечания о глубоких сетях

Сергей Николенко

НИУ ВШЭ – Санкт-Петербург

18 сентября 2020 г.

Random facts:

- 18 сентября 1066 г. Гаральд Гардерада вместе с Тостигом Годвинсоном высадились в устье реки Хамбер; через два дня он разгромил английские войска в битве у Фулфорда, но через пять дней потерпел сокрушительное поражение при Стэмфорд-Бридж
- 18 сентября 1698 г. в Бастилию был переведён таинственный узник под номером 64489001, известный как «Железная маска»; после его смерти в 1703 г. всю мебель и одежду специально уничтожили, закрасив стены и расплавив все металлические вещи
- 18 сентября 1830 г. лошадь победила в гонке на 9 миль от Райлиз Таверн до Балтимора между лошадью и первым американским паровозом, а 18 сентября 1893 г. было завершено строительство Великой Северной железной дороги между Миссисипи и Тихим океаном
- 18 сентября 1870 г. исследователь Генри Уошберн открыл гейзер Old Faithful, который впоследствии стал источником одного из самых известных датасетов для кластеризации
- 18 сентября 1934 г. СССР вошёл в состав Лиги наций; хватило лет на пять

Регуляризация в нейронных сетях

Регуляризация в нейронных сетях

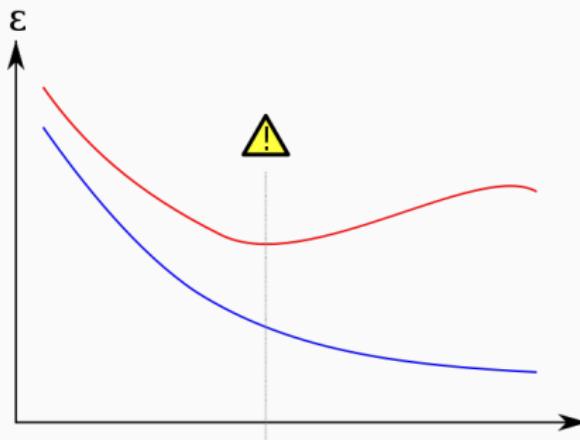
- У нейронных сетей очень много параметров.
- Регуляризация совершенно необходима.
- L_2 или L_1 регуляризация ($\lambda \sum_w w^2$ или $\lambda \sum_w |w|$) — это классический метод и в нейронных сетях, *weight decay*.
- Очень легко добавить: ещё одно слагаемое в целевую функцию, и иногда всё ещё полезно.

Регуляризация в нейронных сетях

- Регуляризация есть во всех библиотеках. Например, в Keras:
 - `W_regularizer` добавит регуляризатор на матрицу весов слоя;
 - `b_regularizer` – на вектор свободных членов;
 - `activity_regularizer` – на вектор выходов.

Регуляризация в нейронных сетях

- Второй способ регуляризации: *ранняя остановка* (early stopping).
- Давайте просто останавливаться, когда начнёт ухудшаться ошибка на валидационном множестве!
- Тоже есть из коробки в *Keras*, через callbacks.



Регуляризация в нейронных сетях

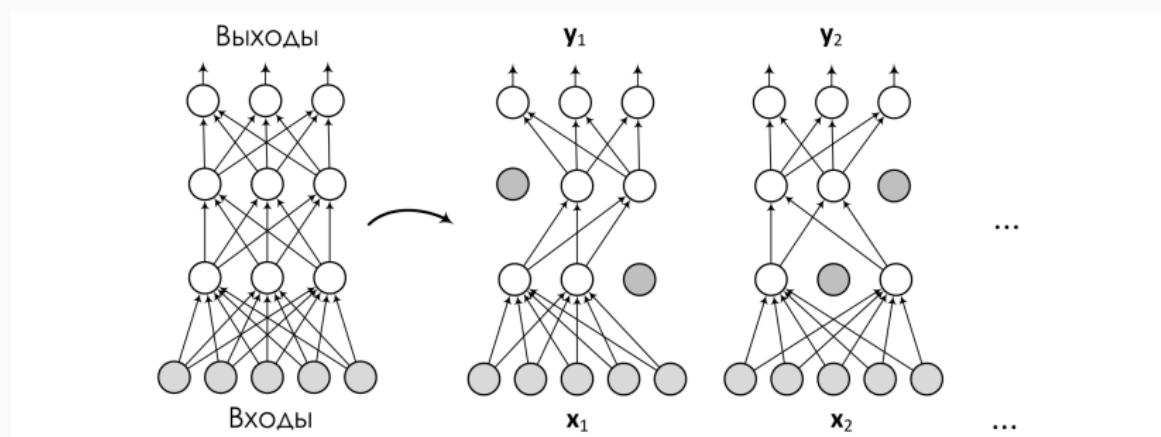
- Третий способ – max-norm constraint.
- Давайте искусственно ограничим норму вектора весов каждого нейрона:

$$\|w\|^2 \leq c.$$

- Это можно делать в процессе оптимизации: когда w выходит за шар радиуса c , проецируем его обратно.

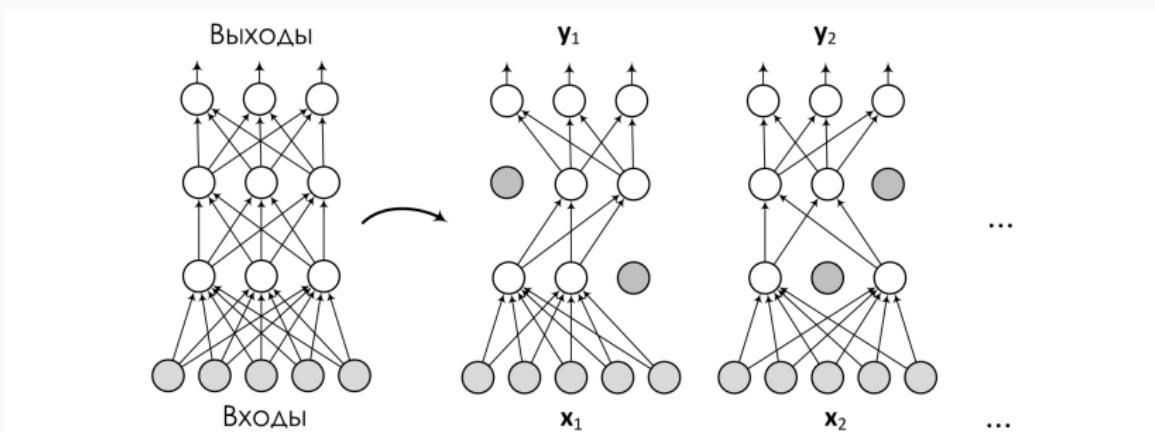
Дропаут

- Но есть и другие варианты.
- Дропаут (dropout): давайте просто выбросим некоторые нейроны случайным образом с вероятностью p !
(Srivastava et al., 2014)



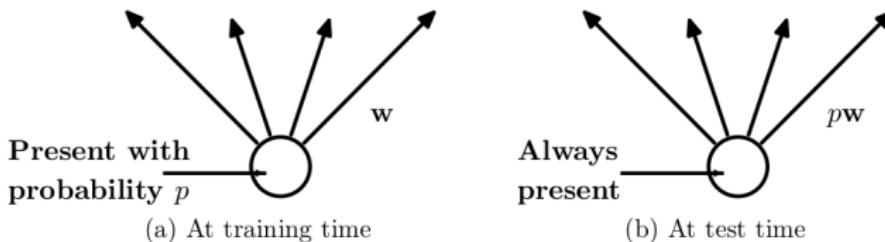
Дропаут

- Получается, что мы сэмплируем кучу сетей, и нейрон получает на вход «среднюю» активацию от разных архитектур.
- Технический вопрос: как потом применять? Неужели надо опять сэмплировать кучу архитектур и усреднять?



Дропаут

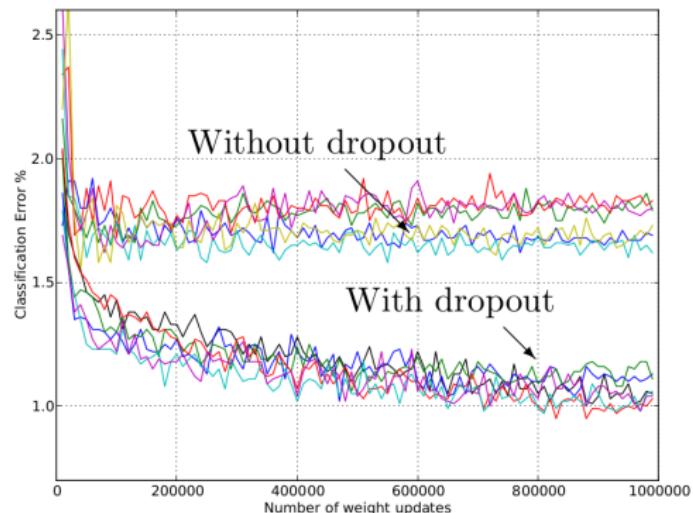
- Чтобы применить обученную сеть, умножим результат на $1/p$, сохраняя ожидание выхода!



- В качестве вероятности часто можно брать просто $p = \frac{1}{2}$, большой разницы нет.

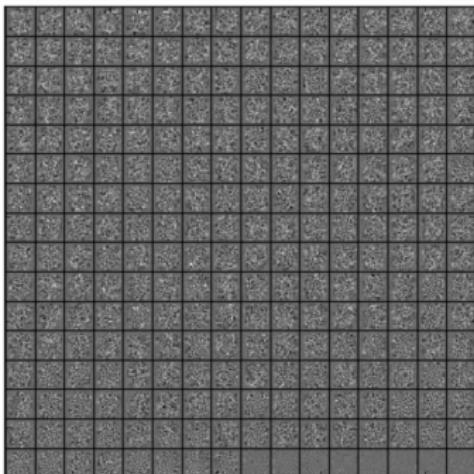
Дропаут

- Dropout улучшает (большие и свёрточные) нейронные сети очень заметно... но почему? WTF?

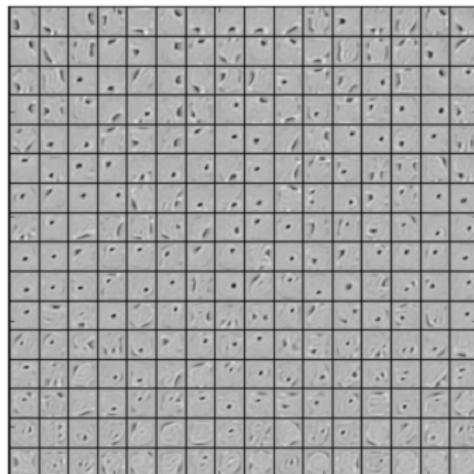


Дропаут

- Идея 1: нейроны теперь должны обучать признаки самостоятельно, а не рассчитывать на других.



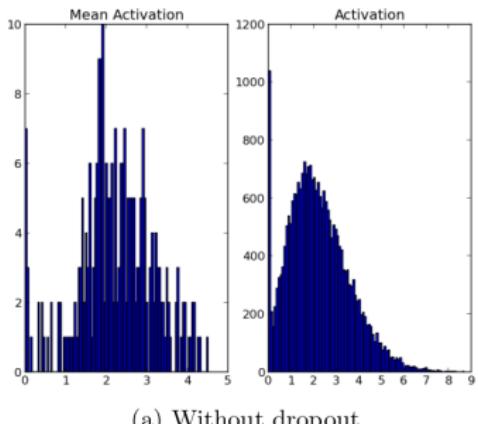
(a) Without dropout



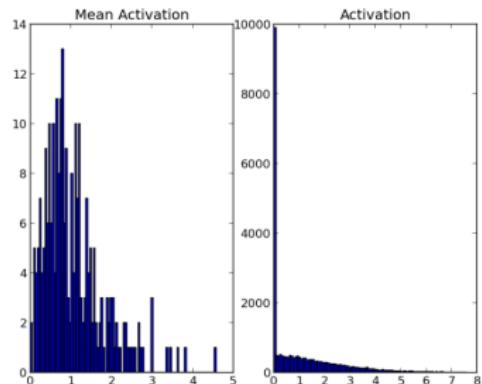
(b) Dropout with $p = 0.5$.

Дропаут

- Аналогично, и разреженность появляется.



(a) Without dropout



(b) Dropout with $p = 0.5$.

Дропаут

- Идея 2: мы как бы усредняем огромное число (2^N) сетей с общими весами, каждую обучая на один шаг.
- Усреднять кучу моделей очень полезно – bootstrapping/xgboost/всё такое...



- Получается, что дропаут – это такой экстремальный бустстреппинг.

- Идея 3: this is just like sex!
- Как работает половое размножение?
- Важно собрать не просто хорошую комбинацию, а устойчивую хорошую комбинацию.

BY ADI LIVNAT AND CHRISTOS PAPADIMITRIOU

Sex as an Algorithm

Дропаут

- Идея 4: нейрон посылает активацию a с вероятностью 0.5.
- Но можно наоборот: давайте посыпать 0.5 (или 1) с вероятностью a .
- Ожидание то же, дисперсия для маленьких p растёт (что неплохо).
- И у нас получаются стохастические нейроны, которые посыпают сигналы случайно – точно как в мозге!
- Улучшение примерно то же, как от dropout, но нужно меньше коммуникации между нейронами (один бит вместо float).
- Т.е. стохастические нейроны в мозге работают как дропаут-регуляризатор!
- Возможно, именно поэтому мы умеем задумываться.

Дропаут

- Идея 5: dropout — это специальная форма априорного распределения.
- Это очень полезный взгляд, с ним победили dropout в рекуррентных сетях.
- Но для этого нужно сначала поговорить о нейронных сетях по-байесовски...
- Вернёмся к этому, если будет время.

Вывод

- Итого получается, что dropout – это очень крутой метод.

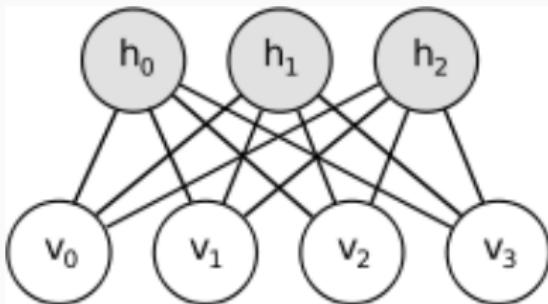


- Но и он сейчас отходит на второй план из-за нормализации по мини-батчам и новых версий градиентного спуска.
- О них чуть позже, а сначала об инициализации весов.

Инициализация весов

Предобучение без учителя

- Революция глубокого обучения началась с *предобучением без учителя* (unsupervised pretraining).
- Главная идея: добраться до хорошей области пространства весов, затем уже сделать fine-tuning градиентным спуском.
- Ограниченные машины Больцмана (restricted Boltzmann machines):



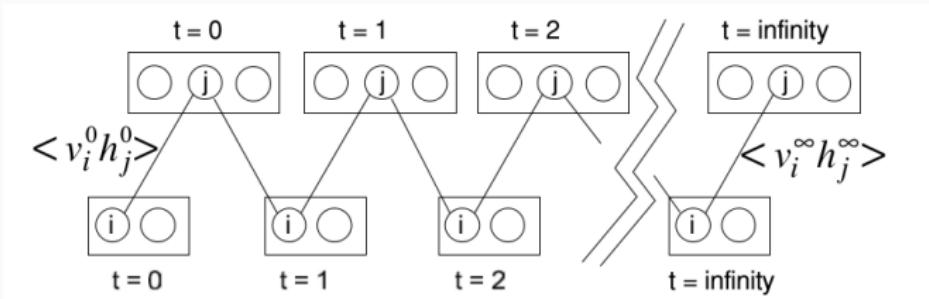
Предобучение без учителя

- Это ненаправленная графическая модель, задающая распределение

$$p(v) = \sum_h p(v, h) = \frac{1}{Z} \sum_h e^{-E(v, h)}, \text{ где}$$

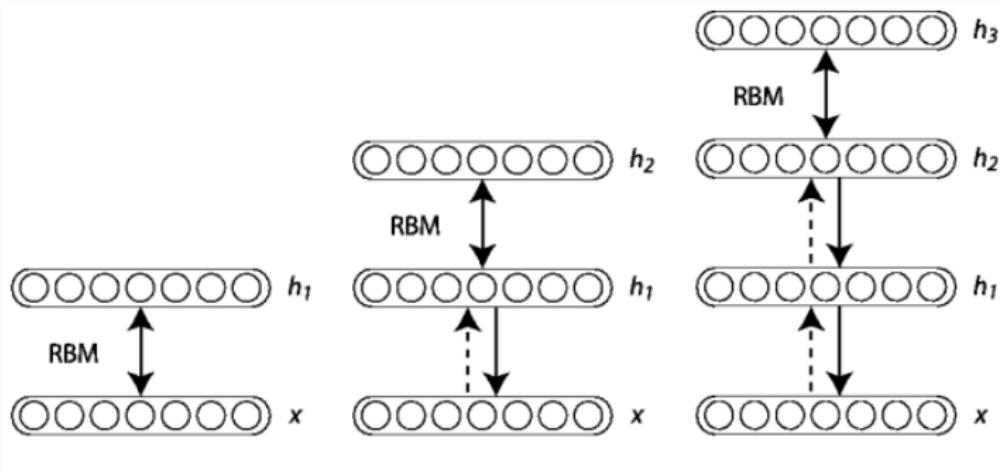
$$E(v, h) = -b^T v - c^T h - h^T W v.$$

- Обучают алгоритмом Contrastive Divergence (приближение к сэмплированию по Гиббсу).



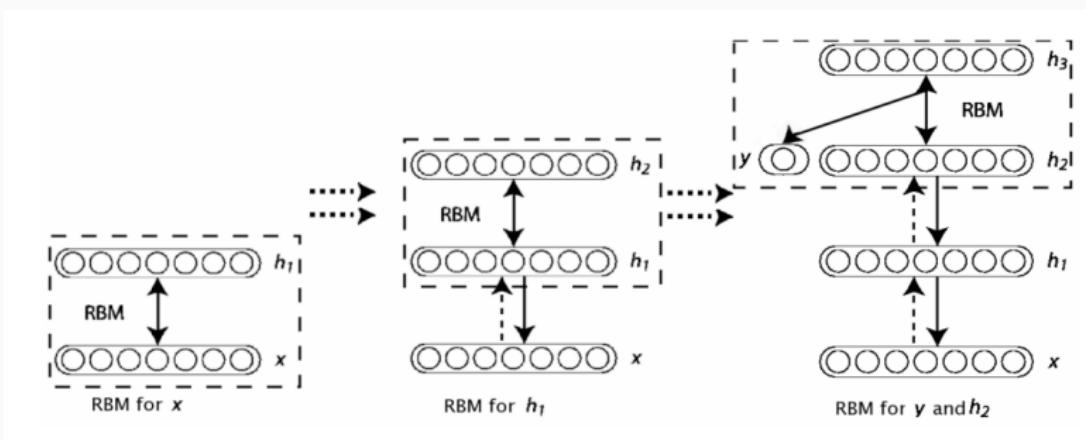
Предобучение без учителя

- Из RBM можно сделать глубокие сети, поставив одну на другую:



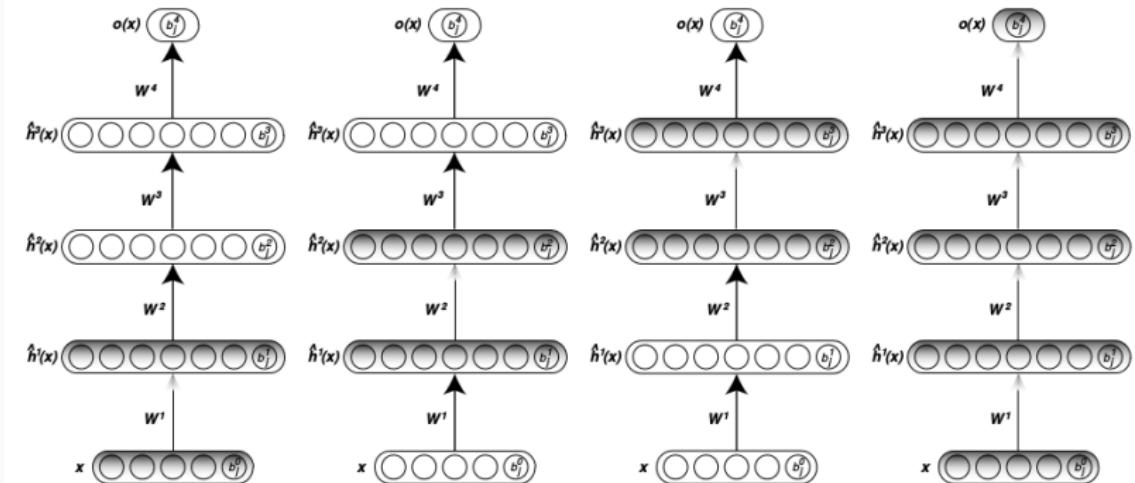
Предобучение без учителя

- И вывод можно вести последовательно, уровень за уровнем:



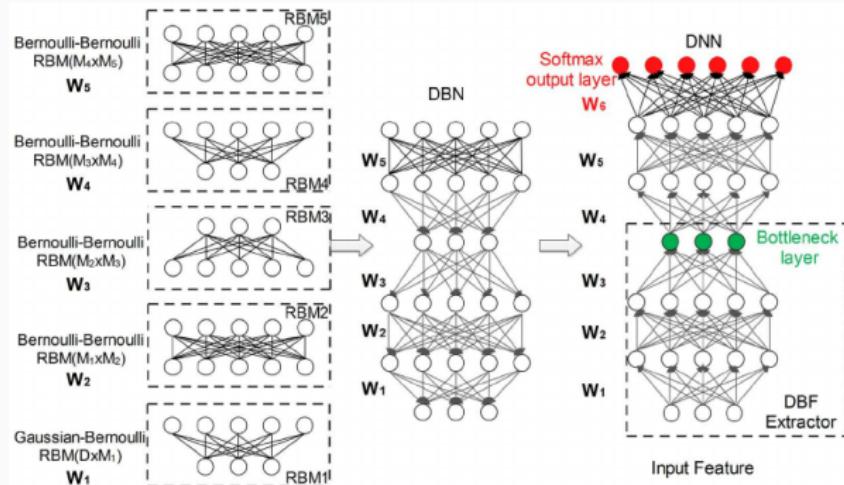
Предобучение без учителя

- А потом уже дообучать градиентным спуском (fine-tuning).



Предобучение без учителя

- Этот подход привёл к прорыву в распознавании речи.



Предобучение без учителя

- Но обучать глубокие сети из RBM довольно сложно, они хрупкие, и вычислительно тоже нелегко.
- И сейчас уже не очень-то и нужны сложные модели вроде RBM для того, чтобы попасть в хорошую начальную область.
- Инициализация весов — важная часть этого.

Инициализация Ксавье

- *Xavier initialization* (Glorot, Bengio, 2010).
- Рассмотрим простой линейный нейрон:

$$y = \mathbf{w}^\top \mathbf{x} + b = \sum_i w_i x_i + b.$$

- Его дисперсия равна

$$\begin{aligned}\text{Var}[y_i] &= \text{Var}[w_i x_i] = \mathbb{E}[w_i^2 x_i^2] - (\mathbb{E}[w_i x_i])^2 = \\ &= \mathbb{E}[x_i]^2 \text{Var}[w_i] + \mathbb{E}[w_i]^2 \text{Var}[x_i] + \text{Var}[w_i] \text{Var}[x_i].\end{aligned}$$

Инициализация Ксавье

- Его дисперсия равна

$$\begin{aligned}\text{Var}[y_i] &= \text{Var}[w_i x_i] = \mathbb{E}[w_i^2 x_i^2] - (\mathbb{E}[w_i x_i])^2 = \\ &= \mathbb{E}[x_i]^2 \text{Var}[w_i] + \mathbb{E}[w_i]^2 \text{Var}[x_i] + \text{Var}[w_i] \text{Var}[x_i].\end{aligned}$$

- Для нулевого среднего весов

$$\text{Var}[y_i] = \text{Var}[w_i] \text{Var}[x_i].$$

- И если w_i и x_i инициализированы независимо из одного и того же распределения,

$$\text{Var}[y] = \text{Var}\left[\sum_{i=1}^{n_{\text{out}}} y_i\right] = \sum_{i=1}^{n_{\text{out}}} \text{Var}[w_i x_i] = n_{\text{out}} \text{Var}[w_i] \text{Var}[x_i].$$

- Иначе говоря, дисперсия на выходе пропорциональна дисперсии на входе с коэффициентом $n_{\text{out}} \text{Var}[w_i]$.

Инициализация Ксавье

- До (Glorot, Bengio, 2010) стандартным способом инициализации было

$$w_i \sim U \left[-\frac{1}{\sqrt{n_{\text{out}}}}, \frac{1}{\sqrt{n_{\text{out}}}} \right].$$

- См., например, *Neural Networks: Tricks of the Trade*.
- Так что с дисперсиями получается

$$\text{Var}[w_i] = \frac{1}{12} \left(\frac{1}{\sqrt{n_{\text{out}}}} + \frac{1}{\sqrt{n_{\text{out}}}} \right)^2 = \frac{1}{3n_{\text{out}}}, \text{ и}$$

$$n_{\text{out}} \text{Var}[w_i] = \frac{1}{3},$$

и после нескольких уровней сигнал совсем умирает; аналогичный эффект происходит и в backprop.

Инициализация Ксавье

- Инициализация Ксавье — давайте попробуем уменьшить изменение дисперсии, т.е. взять

$$\text{Var}[w_i] = \frac{2}{n_{\text{in}} + n_{\text{out}}};$$

для равномерного распределения это

$$w_i \sim U \left[-\frac{\sqrt{6}}{\sqrt{n_{\text{in}} + n_{\text{out}}}}, \frac{\sqrt{6}}{\sqrt{n_{\text{in}} + n_{\text{out}}}} \right].$$

- Но это работает только для симметричных активаций, т.е. не для ReLU...

Инициализация Ксавье

- ...до работы (He et al., 2015). Вернёмся к

$$\text{Var}[w_i x_i] = \mathbb{E}[x_i]^2 \text{Var}[w_i] + \mathbb{E}[w_i]^2 \text{Var}[x_i] + \text{Var}[w_i] \text{Var}[x_i]$$

- Мы теперь можем обнулить только второе слагаемое:

$$\text{Var}[w_i x_i] = \mathbb{E}[x_i]^2 \text{Var}[w_i] + \text{Var}[w_i] \text{Var}[x_i] = \text{Var}[w_i] \mathbb{E}[x_i^2], \text{ и}$$

$$\text{Var}[y^{(l)}] = n_{\text{in}}^{(l)} \text{Var}[w^{(l)}] \mathbb{E}[\left(x^{(l)}\right)^2].$$

Инициализация Ксавье

- Мы теперь можем обнулить только второе слагаемое:

$$\text{Var} \left[y^{(l)} \right] = n_{\text{in}}^{(l)} \text{Var} \left[w^{(l)} \right] \mathbb{E} \left[\left(x^{(l)} \right)^2 \right].$$

- Предположим, что $x^{(l)} = \max(0, y^{(l-1)})$, и у $y^{(l-1)}$ симметричное распределение вокруг нуля. Тогда

$$\mathbb{E} \left[\left(x^{(l)} \right)^2 \right] = \frac{1}{2} \text{Var} \left[y^{(l-1)} \right], \quad \text{Var} \left[y^{(l)} \right] = \frac{n_{\text{in}}^{(l)}}{2} \text{Var} \left[w^{(l)} \right] \text{Var} \left[y^{(l-1)} \right].$$

Инициализация Ксавье

- И это приводит к формуле для дисперсии активации ReLU; теперь нет никакого n_{out} :

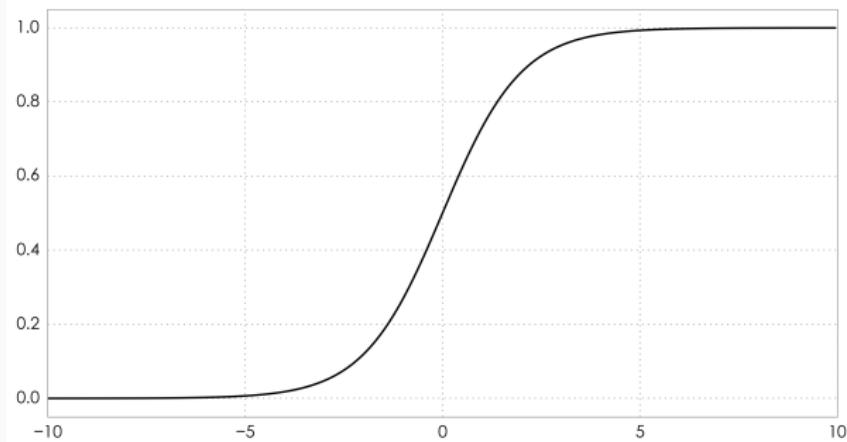
$$\text{Var}[w_i] = 2/n_{\text{in}}^{(l)}.$$

- Кстати, равномерную инициализацию делать не обязательно, можно и нормальное распределение:

$$w_i \sim \mathcal{N} \left(0, \sqrt{2/n_{\text{in}}^{(l)}} \right).$$

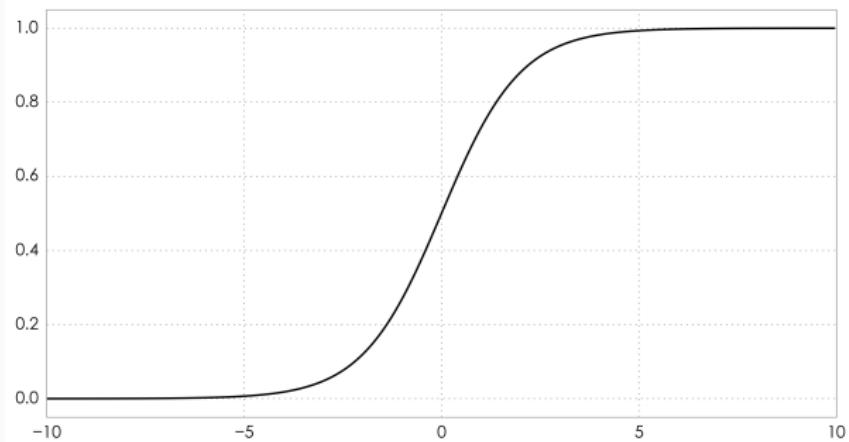
О сигмоидах

- Кстати, о (Glorot, Bengio, 2010) – ещё одна важная идея.
- Эксперименты показали, что $\sigma(x) = \frac{1}{1+e^{-x}}$ работает в глубоких сетях довольно плохо.



О сигмоидах

- Насыщение: если $\sigma(x)$ уже «обучилась», т.е. даёт большие по модулю значения, то её производная близка к нулю и «поменять мнение» трудно.



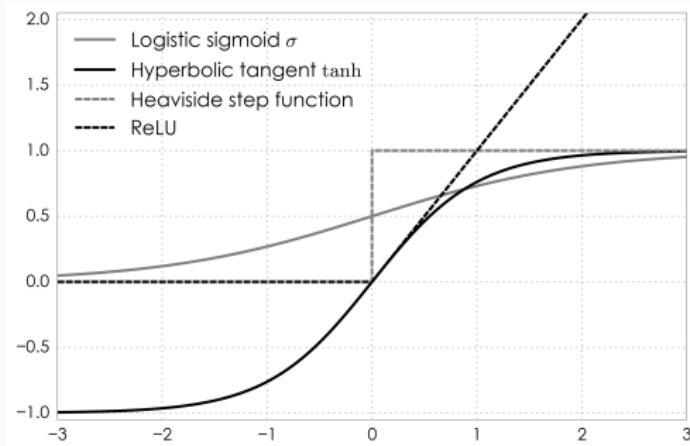
- Но ведь другие тоже насыщаются? В чём разница?

О сигмоидах

- Рассмотрим последний слой сети $h(Wa + b)$, где a — выходы предыдущего слоя, b — свободные члены, h — функция активации последнего уровня, обычно softmax.
- Когда мы начинаем оптимизировать сложную функцию потерь, поначалу выходы h не несут полезной информации о входах, ведь первые уровни ещё не обучены.
- Тогда неплохим приближением будет константная функция, выдающая средние значения выходов.
- Это значит, что $h(Wa + b)$ подберёт подходящие свободные члены b и постарается обнулить слагаемое Wh , которое поначалу скорее шум, чем сигнал.

О сигмоидах

- Иначе говоря, в процессе обучения мы постараемся привести выходы предыдущего слоя к нулю.
- Здесь и проявляется разница: $y = \sigma(x) = \frac{1}{1+e^{-x}}$ область значений $(0, 1)$ при среднем $\frac{1}{2}$, и при $\sigma(x) \rightarrow 0$ будет и $\sigma'(x) \rightarrow 0$.
- А у \tanh наоборот: когда $\tanh(x) \rightarrow 0$, $\tanh'(x)$ максимальна.



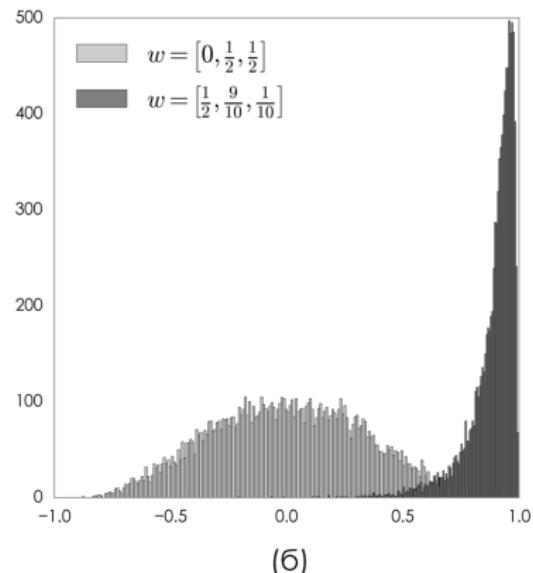
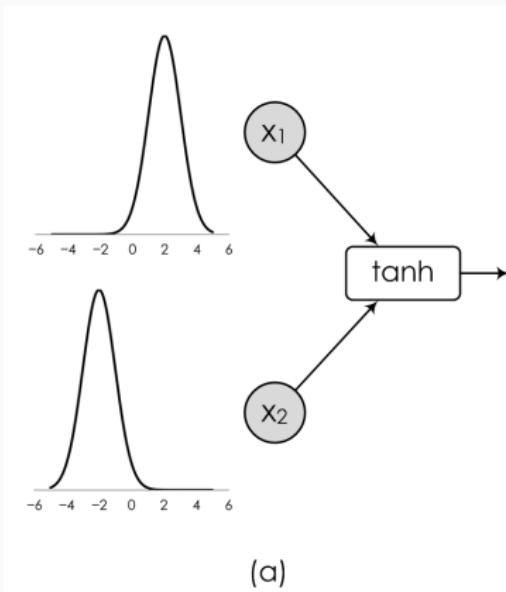
Нормализация по мини-батчам

Нормализация по мини-батчам

- Ещё одна важная проблема в глубоких сетях: *внутренний сдвиг переменных* (internal covariate shift).
- Когда меняются веса слоя, меняется распределение его выходов.
- Это значит, что следующему уровню придётся всё начинать заново, он же не ожидал таких входов, не видел их раньше!
- Более того, нейроны следующего уровня могли уже и насытиться, и им теперь сложно быстро обучиться заново.
- Это серьёзно мешает обучению.

Нормализация по мини-батчам

- Вот характерный пример:



- Что делать?

Нормализация по мини-батчам

- Можно пытаться нормализовать входы каждого уровня.
- Не работает: рассмотрим для простоты уровень с одним только bias b и входами u :

$$\hat{x} = x - \mathbb{E}[x], \text{ где } x = u + b.$$

- На следующем шаге градиентного спуска получится $b := b + \Delta b\dots$
- ...но \hat{x} не изменится:

$$u + b + \Delta b - \mathbb{E}[u + b + \Delta b] = u + b - \mathbb{E}[u + b].$$

- Так что всё обучение сведётся к тому, что b будет неограниченно расти — не очень хорошо.

Нормализация по мини-батчам

- Можно пытаться добавить нормализацию как отдельный слой:

$$\hat{\mathbf{x}} = \text{Norm}(\mathbf{x}, \mathcal{X}).$$

- Это лучше, но теперь этому слою на входе нужен весь датасет \mathcal{X} !
- И на шаге градиентного спуска придётся вычислить $\frac{\partial \text{Norm}}{\partial \mathbf{x}}$ и $\frac{\partial \text{Norm}}{\partial \mathcal{X}}$, да ещё и матрицу ковариаций

$$\text{Cov}[\mathbf{x}] = \mathbb{E}_{\mathbf{x} \in \mathcal{X}} [\mathbf{x}\mathbf{x}^\top] - \mathbb{E}[\mathbf{x}] \mathbb{E}[\mathbf{x}]^\top.$$

- Это точно не сработает.

Нормализация по мини-батчам

- Решение в том, чтобы нормализовать каждый вход отдельно, и не по всему датасету, а по текущему кусочку; это и есть *нормализация по мини-батчам* (batch normalization).
- После нормализации по мини-батчам получим

$$\hat{x}_k = \frac{x_k - \mathbb{E}[x_k]}{\sqrt{\text{Var}[x_k]}},$$

где статистики подсчитаны по текущему мини-батчу.

- Ещё одна проблема: теперь пропадают нелинейности!
- Например, σ теперь практически всегда близка к линейной.

Нормализация по мини-батчам

- Чтобы это исправить, нужно добавить гибкости уровню batchnorm.
- В частности, нужно разрешить ему обучаться иногда *ничего не делать* со входами.
- Так что вводим дополнительные параметры (сдвиг и растяжение):

$$y_k = \gamma_k \hat{x}_k + \beta_k = \gamma_k \frac{x_k - \mathbb{E}[x_k]}{\sqrt{\text{Var}[x_k]}} + \beta_k.$$

- γ_k и β_k — это новые переменные, тоже будут обучаться градиентным спуском, как веса.

Нормализация по мини-батчам

- И ещё добавим ϵ в знаменатель, чтобы на ноль не делить.
- Теперь мы можем формально описать слой батч-нормализации — для очередного мини-батча $B = \{x_1, \dots, x_m\}$:
 - вычислить базовые статистики по мини-батчу

$$\mu_B = \frac{1}{m} \sum_{i=1}^m x_i, \quad \sigma_B^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2,$$

- нормализовать входы

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}},$$

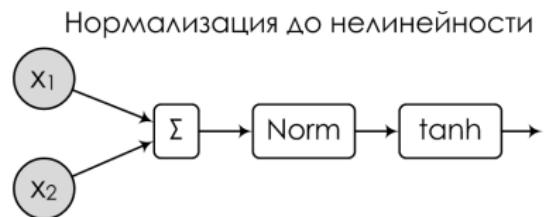
- вычислить результат

$$y_i = \gamma x_i + \beta.$$

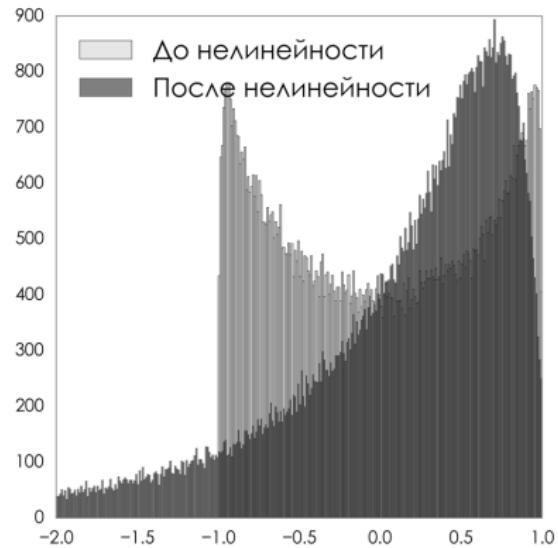
- Через всё это совершенно стандартным образом пропускаются градиенты, в том числе по γ и β .

Нормализация по мини-батчам

- Последнее замечание: важно, куда поместить слой batchnorm.
- Можно до, а можно после нелинейности.



(а)



Варианты

- Нормализация по мини-батчам сейчас стала фактически стандартом.
- Очередная очень крутая история, примерно как дропаут.
- Но мысль идёт и дальше:
 - (Laurent et al., 2016): BN не помогает рекуррентным сетям;
 - (Cooijmans et al., 2016): recurrent batch normalization;
 - (Salimans and Kingma, 2016): нормализация весов для улучшения обучения — давайте добавим веса как

$$h_i = f \left(\frac{\gamma}{\|w_i\|} w_i^\top x + b_i \right);$$

тогда мы будем перемасштабировать градиент, стабилизировать его норму и приближать его матрицу ковариаций к единичной, что улучшает обучение.

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):

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

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

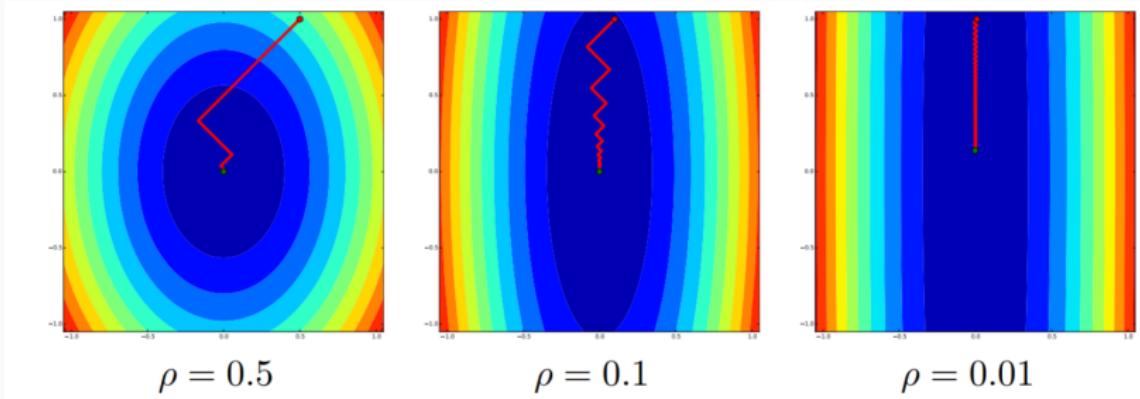
$$|\phi'_k(\alpha)| \leq c_2 |\phi'_k(0)|,$$

т.е. мы хотим уменьшить проекцию градиента.

- Останавливаем когда $\|\nabla_{\mathbf{x}} f(\mathbf{x}_k)\|^2 \leq \epsilon$ или $\|\nabla_{\mathbf{x}} f(\mathbf{x}_k)\|^2 \leq \epsilon \|\nabla_{\mathbf{x}} f(\mathbf{x}_0)\|^2$ (а почему квадрат, кстати?).

Градиентный спуск

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



- 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

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

- Здесь тоже применимо условие Армихо:

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

- Было бы круто! Но H_k посчитать просто нереально.

Градиентный спуск

- Есть, правда, приближения.
- Метод сопряжённых градиентов, квази-ニュтоновские 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*:

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

- минимизация эмпирического риска

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

- минимизация вариационной нижней оценки (ELBO)... но об этом позже.
- Что такое теперь, получается, мини-батчи?

Стохастический градиентный спуск

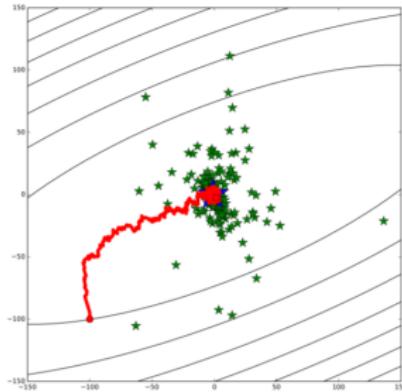
- Да просто эмпирические оценки общей функции по подвыборке:

$$\hat{F}(x) = \frac{1}{m} \sum_{i=1}^m f(x, y_i), \quad \hat{g}(x) = \frac{1}{m} \sum_{i=1}^m \nabla_x f(x, y_i).$$

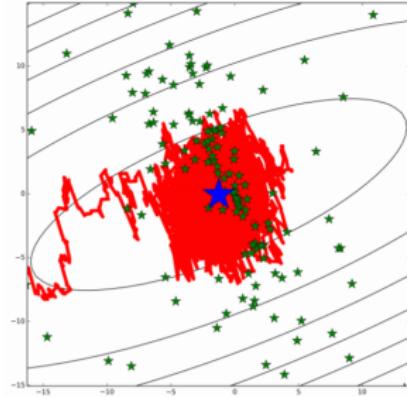
- Это очень хорошие оценки: несмешённые, сходятся на бесконечности (правда, медленно), легко посчитать.
- В целом, так и мотивируется стохастический градиентный спуск (SGD): метод Монте-Карло по сути.
- Но есть проблемы...

Стохастический градиентный спуск

- Проблемы SGD:
 - никогда не идёт в правильном направлении,
 - шаг не равен нулю в оптимуме $F(x)$, т.е. не может сойтись с постоянной длиной шага,
 - мы не знаем ни $F(x)$, ни $\nabla F(x)$, 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 $F(\mathbf{x}) = \mathbb{E}_{q(y)} f(\mathbf{x}, y) \rightarrow \min_{\mathbf{x}}$:

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

- Давайте оценим невязку точки на очередной итерации:

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

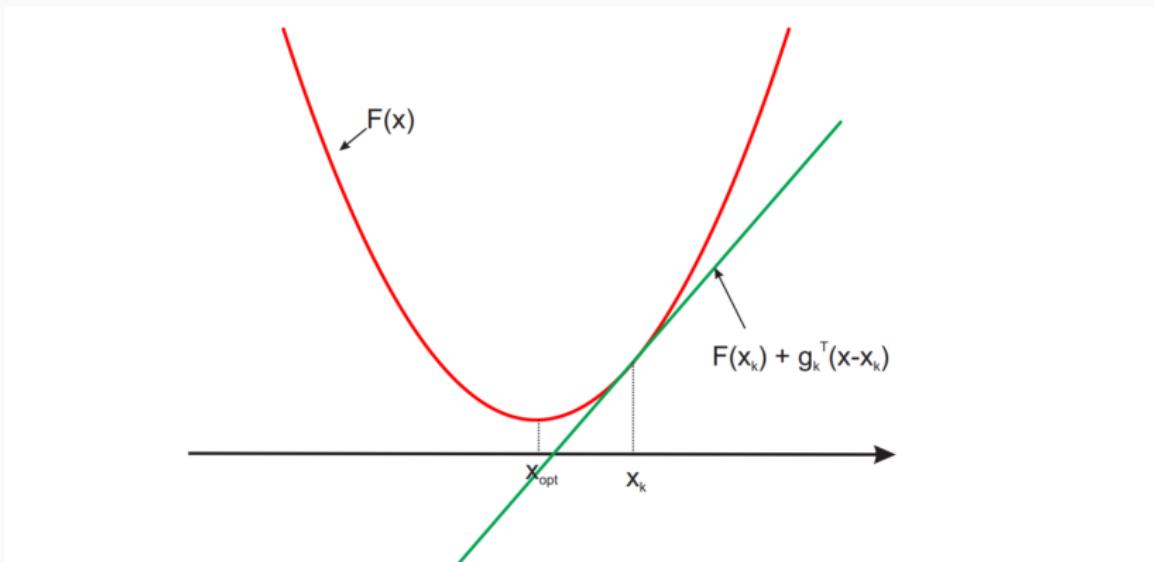
- Возьмём ожидание по $q(y)$ в момент времени k :

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

Стохастический градиентный спуск

- Для простоты предположим, что F выпуклая:

$$F(x_{\text{opt}}) \geq F(x_k) + g_k^\top (x_k - x_{\text{opt}})$$



Стохастический градиентный спуск

- У нас было

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

- Значит,

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

Стохастический градиентный спуск

- Возьмём ожидание от левой части и просуммируем:

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

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

Стохастический градиентный спуск

- Воспользуемся выпуклостью:

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

- 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 $\alpha_i = h$

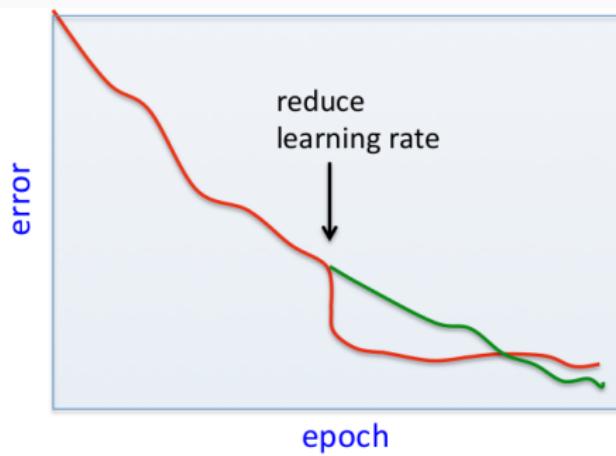
$$\mathbb{E}F(\hat{\mathbf{x}}_k) - F(\mathbf{x}_{\text{opt}}) \leq \frac{R^2}{2h(k+1)} + \frac{G^2h}{2} \xrightarrow{k \rightarrow \infty} \frac{G^2h}{2}.$$

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 идеи 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.

Метод моментов

- Значит, нужны какие-то улучшения. Что-то делать со скоростью обучения.
- Скорость обучения лучше не уменьшать слишком быстро.



- Но это в любом случае никак не учитывает собственно F .

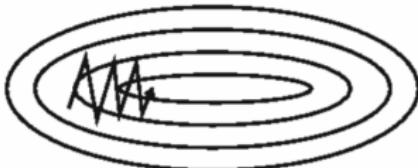
Метод моментов

- *Метод моментов* (momentum): сохраним часть скорости, как у материальной точки.
- С инерцией получается

$$u_t = \gamma u_{t-1} + \eta \nabla_x F(x),$$

$$x = x - u_t.$$

- И теперь мы сохраняем γu_{t-1} .



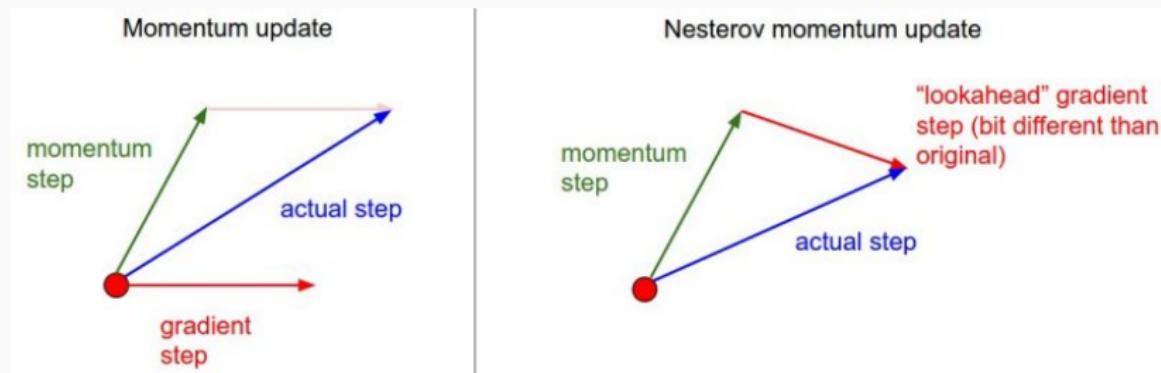
without momentum



with momentum

Метод моментов

- Мы ведь на самом деле уже знаем, что попадём в γu_{t-1} на промежуточном шаге.
- Давайте прямо там, на полпути, и вычислим градиент!



Метод моментов

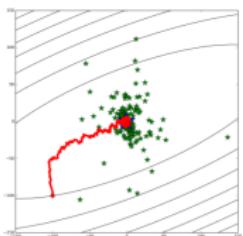
- *Метод Нестерова* (Nesterov's momentum):

$$u_t = \gamma u_{t-1} + \eta \nabla_x F(x - \gamma u_{t-1})$$

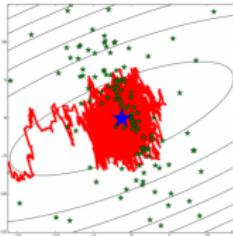


Метод моментов

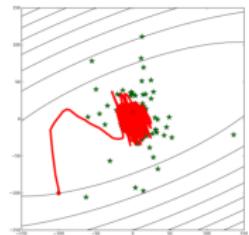
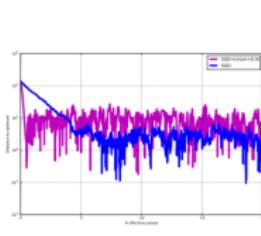
- Всё равно, конечно, проблемы не пропадают:



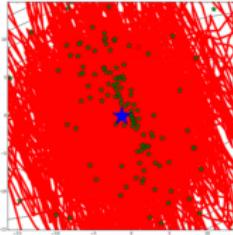
SGD



SGD in opt. vicinity



SGD+mom.



SGD+mom. in opt.

- Можно ли ещё лучше?..

Адаптивные методы градиентного спуска

- Заметим, что до сих пор скорость обучения была одна во всех направлениях, мы пытались выбрать направление как бы глобально.
- Идея: давайте быстрее двигаться по тем параметрам, которые не сильно меняются, и медленнее по быстро меняющимся параметрам.

Адаптивные методы градиентного спуска

- *Adagrad*: давайте накапливать историю этой скорости изменений и учитывать её.
- Обозначая $g_{t,i} = \nabla_{w_i} L(w)$, получим

$$w_{t+1,i} = w_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i},$$

где G_t – диагональная матрица с $G_{t,ii} = G_{t-1,ii} + g_{t,i}^2$, которая накапливает общее значение градиента по всей истории обучения.

- Так что скорость обучения всё время уменьшается, но с разной скоростью для разных w_i .

Адаптивные методы градиентного спуска

- Проблема: G всё увеличивается и увеличивается, и скорость обучения иногда уменьшается слишком быстро.
- *Adadelta* (Zeiler, 2012) – та же идея, но две новых модификации.
- Во-первых, историю градиентов мы теперь считаем с затуханием:

$$G_{t,ii} = \rho G_{t-1,ii} + (1 - \rho) g_{t,i}^2.$$

- А всё остальное здесь точно так же:

$$u_t = -\frac{\eta}{\sqrt{G_{t-1} + \epsilon}} \mathbf{g}_{t-1}.$$

Адаптивные методы градиентного спуска

- Во-вторых, надо бы «единицы измерения» привести в соответствие.
- В предыдущих методах была проблема:
 - в обычном градиентном спуске или методе моментов «единицы измерения» обновления параметров Δw — это единицы измерения градиента, т.е. если веса в секундах, а целевая функция в метрах, то градиент будет иметь размерность «метр в секунду», и мы вычитаем метры в секунду из секунд;
 - а в Adagrad получалось, что значения обновлений Δw зависели от отношений градиентов, и величина обновлений вовсе безразмерная.

Адаптивные методы градиентного спуска

- Эта проблема решается в методе второго порядка:
обновление параметров Δw пропорционально $H^{-1}\nabla_w f$, то есть размерность будет

$$\Delta w \propto H^{-1}\nabla_w f \propto \frac{\frac{\partial f}{\partial w}}{\frac{\partial^2 f}{\partial w^2}} \propto \text{размерность } w.$$

- Чтобы привести *Adadelta* в соответствие, нужно домножить на ещё одно экспоненциальное среднее, но теперь уже от квадратов обновлений параметров, а не от градиента.
- Настоящее среднее мы не знаем, аппроксимируем предыдущими шагами:

$$\mathbb{E} [\Delta w^2]_t = \rho \mathbb{E} [\Delta w^2]_{t-1} + (1 - \rho) \Delta w^2, \text{ где}$$

$$u_t = -\frac{\sqrt{\mathbb{E} [\Delta w^2]_{t-1} + \epsilon}}{\sqrt{G_{t-1} + \epsilon}} \cdot g_{t-1}.$$

Адаптивные методы градиентного спуска

- Следующий вариант – *RMSprop* из курса Хинтона.
- Практически то же, что *Adadelta*, только *RMSprop* не делает вторую поправку с изменением единиц и хранением истории самих обновлений, а просто использует корень из среднего от квадратов (вот он где, RMS) от градиентов:

$$u_t = -\frac{\eta}{\sqrt{G_{t-1} + \epsilon}} \cdot g_{t-1}.$$

Адаптивные методы градиентного спуска

- И последний алгоритм – *Adam* (Kingma, Ba, 2014).
- Модификация *Adagrad* со сглаженными версиями среднего и среднеквадратичного градиентов:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t,$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2,$$

$$u_t = \frac{\eta}{\sqrt{v + \epsilon}} m_t.$$

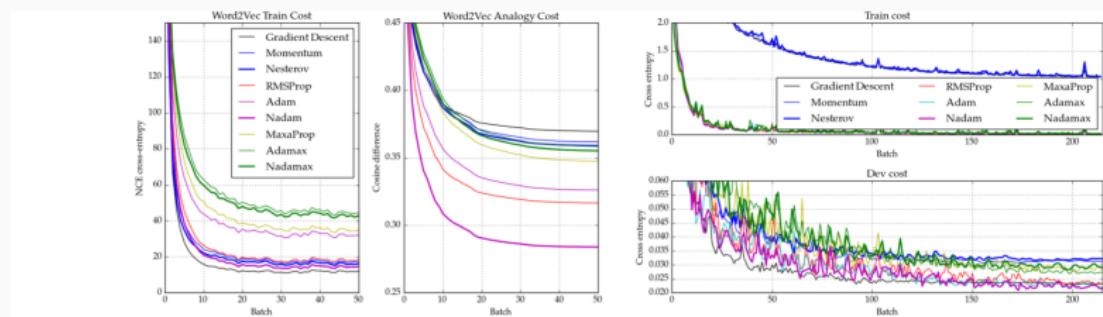
- (Kingma, Ba, 2014) рекомендуют $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$.
- *Adam* практически не требует настройки, используется на практике очень часто.
- ...[<http://ruder.io/optimizing-gradient-descent/>]

Адаптивные методы градиентного спуска

- А ещё можно совместить Adam и Нестерова – Nadam (Dozat, 2016)

Algorithm 8 Nesterov-accelerated adaptive moment estimation

$$\begin{aligned}\mathbf{g}_t &\leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1}) \\ \hat{\mathbf{g}} &\leftarrow \frac{\mathbf{g}_t}{1 - \prod_{i=1}^t \mu_i} \\ \mathbf{m}_t &\leftarrow \mu \mathbf{m}_{t-1} + (1 - \mu) \mathbf{g}_t \\ \hat{\mathbf{m}}_t &\leftarrow \frac{\mathbf{m}_t}{1 - \prod_{i=1}^{t-1} \mu_i} \\ \mathbf{n}_t &\leftarrow v \mathbf{n}_{t-1} + (1 - v) \mathbf{g}_t^2 \\ \hat{\mathbf{n}}_t &\leftarrow \frac{\mathbf{n}_t}{1 - v^t} \\ \tilde{\mathbf{m}}_t &\leftarrow (1 - \mu_t) \hat{\mathbf{g}}_t + \mu_{t+1} \hat{\mathbf{m}}_t \\ \theta_t &\leftarrow \theta_{t-1} - \eta \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{n}}_t} + \epsilon}\end{aligned}$$



Адаптивные методы градиентного спуска

- Чуть более общий взгляд – всё это выглядит вот так:
 - обычный стохастический градиентный спуск: $w_{k+1} = w_k - \alpha_k \tilde{\nabla}f(w_k)$, где $\tilde{\nabla}f(w_k) = \nabla f(w_k; x_{i_k})$;
 - SGD с моментами: $w_{k+1} = w_k - \alpha_k \tilde{\nabla}f(w_k + \gamma_k (w_k - w_{k-1})) + \beta_k (w_k - w_{k-1})$;
 - адаптивный SGD:

$$w_{k+1} = w_k - \alpha_k H_k^{-1} \tilde{\nabla}f(w_k + \gamma_k (w_k - w_{k-1})) + \beta_k H_k^{-1} H_{k-1} (w_k - w_{k-1}),$$

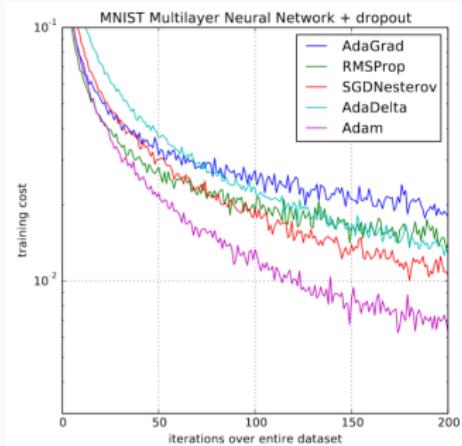
где обычно $H_k = \text{diag} \left(\left[\sum_{i=1}^k \eta_i g_i \circ g_i \right]^{1/2} \right)$, где

$g_k = \tilde{\nabla}f(w_k + \gamma_k (w_k - w_{k-1}))$ (т.е. H_k – это диагональная матрица, элементы которой – квадратные корни из линейных комбинаций квадратов предыдущих градиентов).

- Т.е. адаптивные методы пытаются подстроиться под геометрию в пространстве данных, а SGD и его варианты используют базовую L_2 -геометрию с $H_k = I$.

Adam, AdamW и другие животные

- Когда Adam появился, все были очень счастливы, и было отчего:

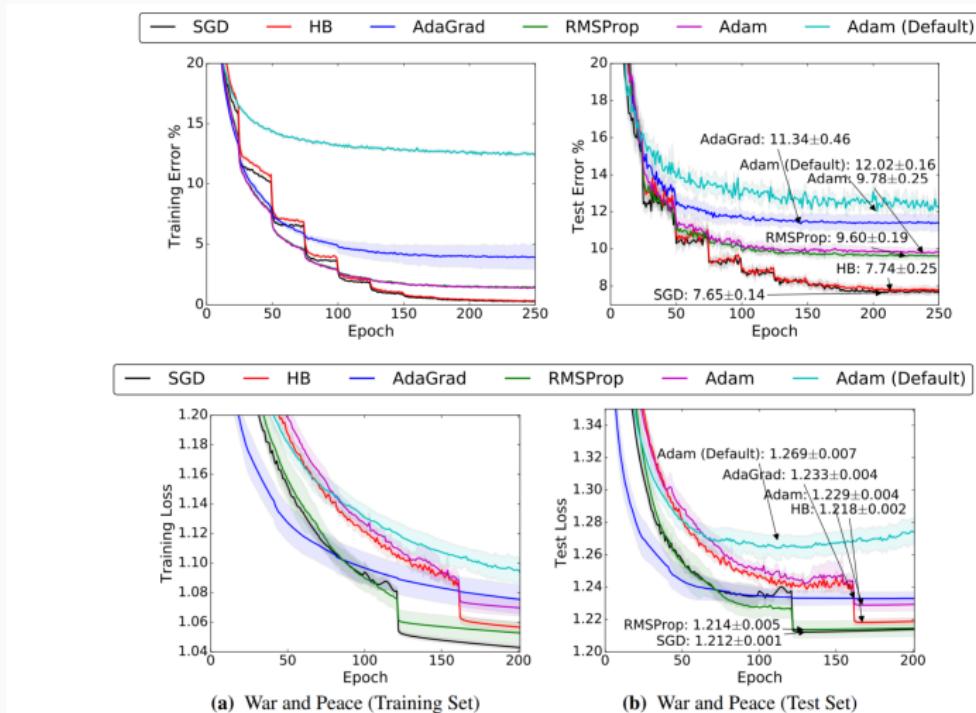


- Но потом выяснилось, что не всё так просто... часто применяли простой SGD. Почему?

- The Marginal Value of Adaptive Gradient Methods in Machine Learning (Wilson et al., May 2017)
- Основные выводы:
 - когда в задаче несколько глобальных минимумов, разные алгоритмы могут найти совершенно разные решения из одной и той же начальной точки;
 - в частности, адаптивные методы могут приходить к оверфиттингу, т.е. не-обобщающимся локальным решениям;
 - и это *не только теоретический худший случай, но и практика.*

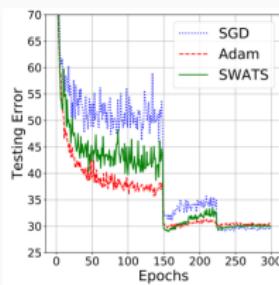
Adam, AdamW и другие животные

- The Marginal Value of Adaptive Gradient Methods in Machine Learning (Wilson et al., May 2017)

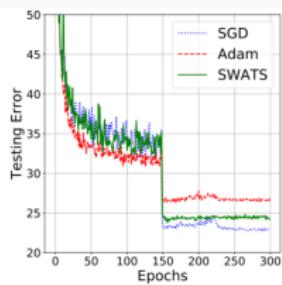


Adam, AdamW и другие животные

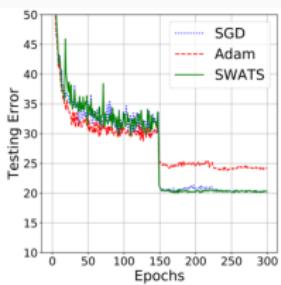
- Но Adam всё равно гораздо быстрее начинает, конечно.
- Поэтому предлагали, например, переключаться с Adam на SGD в нужное время (Keskar, Socher, Dec 2017)



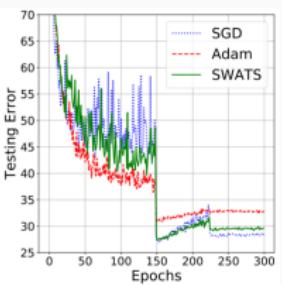
(e) ResNet-32 — CIFAR-100



(f) DenseNet — CIFAR-100



(g) PyramidNet — CIFAR-100



(h) SENet — CIFAR-100

- Всё равно, конечно, SGD не получилось превзойти, и даже не всегда наравне.
- Но и это ещё не вся история...

- Fixing Weight Decay Regularization in Adam (Loshchilov, Hutter, Feb 2018):
 - я мельком говорил, что weight decay – это то же самое, что L_2 -регуляризация;
 - но для адаптивных методов это, оказывается, не совсем так...
 - давайте разберёмся, что такое weight decay и почему это может быть не эквивалентно.

Adam, AdamW и другие животные

- Исходный weight decay (Hanson, Pratt, 1988):

$$\mathbf{x}_{t+1} = (1 - w)\mathbf{x}_t - \eta \nabla f_t(\mathbf{x}_t),$$

где w – это скорость weight decay, η – скорость обучения.

- Там же сразу отмечено, что это эквивалентно тому, чтобы поменять f :

$$f_t^{\text{reg}}(\mathbf{x}_t) = f_t(\mathbf{x}_t) + \frac{w}{2} \|\mathbf{x}_t\|_2^2.$$

- А можно и просто градиент подправить:

$$\nabla f_t^{\text{reg}}(\mathbf{x}_t) = \nabla f_t(\mathbf{x}_t) + w\mathbf{x}_t.$$

- Это всё, конечно, верно, но...

Adam, AdamW и другие животные

- ...но не работает уже даже просто с моментами:

Algorithm 1 SGD with L₂ regularization and
SGD with weight decay (SGDW), both with momentum

- 1: **given** initial learning rate $\alpha \in \mathbb{R}$, momentum factor $\beta_1 \in \mathbb{R}$, weight decay / L₂ regularization factor $w \in \mathbb{R}$
 - 2: **initialize** time step $t \leftarrow 0$, parameter vector $\mathbf{x}_{t=0} \in \mathbb{R}^n$, first moment vector $\mathbf{m}_{t=0} \leftarrow \mathbf{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
 - 3: **repeat**
 - 4: $t \leftarrow t + 1$
 - 5: $\nabla f_t(\mathbf{x}_{t-1}) \leftarrow \text{SelectBatch}(\mathbf{x}_{t-1})$ ▷ select batch and return the corresponding gradient
 - 6: $\mathbf{g}_t \leftarrow \nabla f_t(\mathbf{x}_{t-1}) + w\mathbf{x}_{t-1}$
 - 7: $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, be used for warm restarts
 - 8: $\mathbf{m}_t \leftarrow \beta_1 \mathbf{m}_{t-1} + \eta_t \alpha \mathbf{g}_t$
 - 9: $\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \mathbf{m}_t - \eta_t w \mathbf{x}_{t-1}$
 - 10: **until** stopping criterion is met
 - 11: **return** optimized parameters \mathbf{x}_t
-

- Получается, что \mathbf{x}_t затухает на $\alpha w \mathbf{x}_{t-1}$, а не на $w \mathbf{x}_{t-1}$; чтобы восстановить поведение, надо взять $w_t = \frac{\alpha}{\alpha'} \delta$, и теперь выбор гиперпараметров α и w завязан друг на друга.
- Решение – просто перенести decay в само изменение \mathbf{x}_t (строка 9) и добавить масштабирование η_t .

Adam, AdamW и другие животные

- То же самое происходит и с Adam:

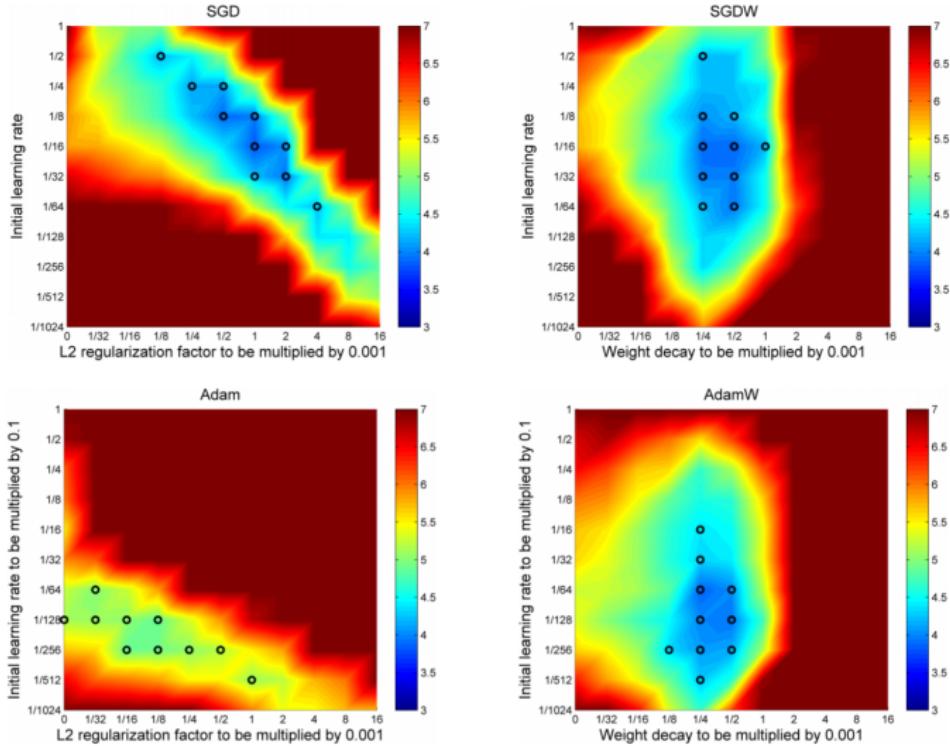
Algorithm 2 Adam with L₂ regularization and
Adam with weight decay (AdamW)

```
1: given  $\alpha = 0.001$ ,  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$ ,  $\epsilon = 10^{-8}$ ,  $w \in \mathbb{R}$ 
2: initialize time step  $t \leftarrow 0$ , parameter vector  $\mathbf{x}_{t=0} \in \mathbb{R}^n$ , first
   moment vector  $\mathbf{m}_{t=0} \leftarrow \mathbf{0}$ , second moment vector  $\mathbf{v}_{t=0} \leftarrow \mathbf{0}$ ,
   schedule multiplier  $\eta_{t=0} \in \mathbb{R}$ 
3: repeat
4:    $t \leftarrow t + 1$ 
5:    $\nabla f_t(\mathbf{x}_{t-1}) \leftarrow \text{SelectBatch}(\mathbf{x}_{t-1})$       ▷ select batch and
   return the corresponding gradient
6:    $\mathbf{g}_t \leftarrow \nabla f_t(\mathbf{x}_{t-1}) + w\mathbf{x}_{t-1}$ 
7:    $\mathbf{m}_t \leftarrow \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t$       ▷ here and below all
   operations are element-wise
8:    $\mathbf{v}_t \leftarrow \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) \mathbf{g}_t^2$ 
9:    $\hat{\mathbf{m}}_t \leftarrow \mathbf{m}_t / (1 - \beta_1^t)$       ▷  $\beta_1$  is taken to the power of  $t$ 
10:   $\hat{\mathbf{v}}_t \leftarrow \mathbf{v}_t / (1 - \beta_2^t)$       ▷  $\beta_2$  is taken to the power of  $t$ 
11:   $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$  ▷ can be fixed, decay, or
   also be used for warm restarts
12:   $\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \eta_t \left( \hat{\mathbf{m}}_t / (\sqrt{\hat{\mathbf{v}}_t} + \epsilon) + w\mathbf{x}_{t-1} \right)$ 
13: until stopping criterion is met
14: return optimized parameters  $\mathbf{x}_t$ 
```

- В базовом Adam веса с большими градиентами меньше затухают, что не всегда хорошо; AdamW восстанавливает исходное поведение.

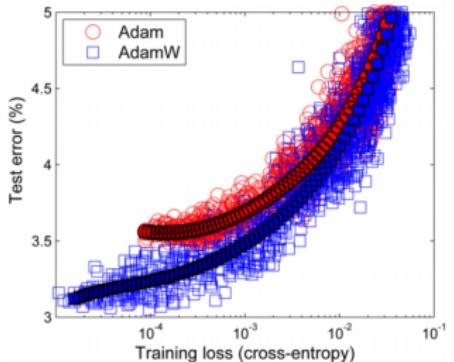
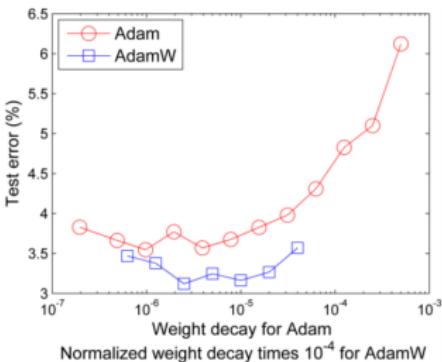
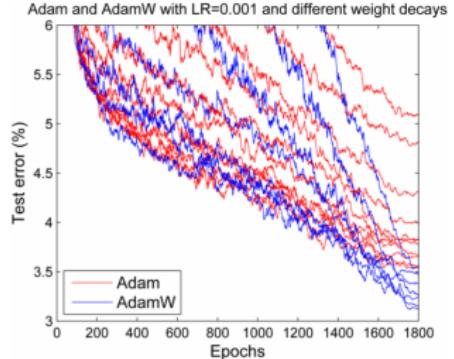
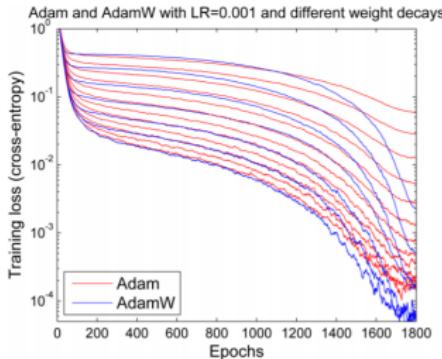
Adam, AdamW и другие животные

- И теперь гиперпараметры разделяются лучше:



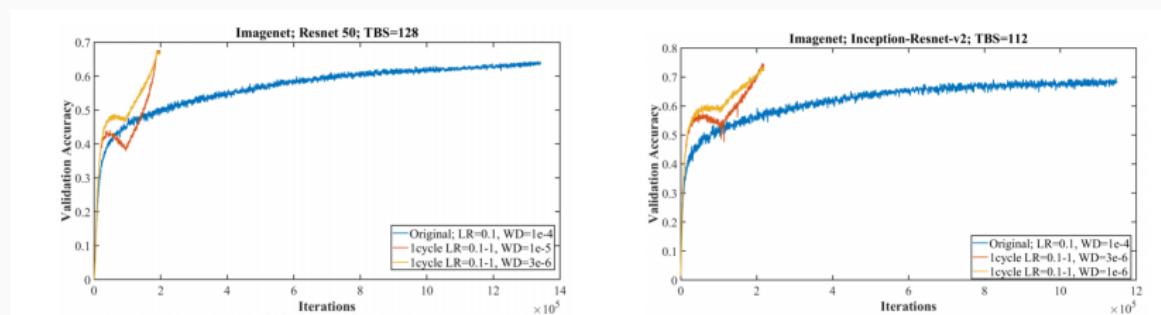
Adam, AdamW и другие животные

- Да и обобщается лучше:



Super-Convergence и amsgrad

- И есть две мысли. Первая – Super-Convergence (Smith, Topin, Aug 2017)
- Идея в том, чтобы менять скорость обучения циклически –
- Это по сути смесь curriculum learning (Bengio et al., 2009) и классического simulated annealing.



- Но вроде бы это не воспроизводится устойчиво.

Super-Convergence и amsgrad

- Другая идея с похожей судьбой – amsgrad (Reddi et al., Mar 2018), ICLR 2018 best paper! Идея:
 - экспоненциальное среднее в Adam/RMSprop может привести к не-сходимости;
 - в частности, в доказательстве сходимости Adam есть ошибка;
 - а AMSGrad хранит максимальный размер градиента, и это типа лучше.

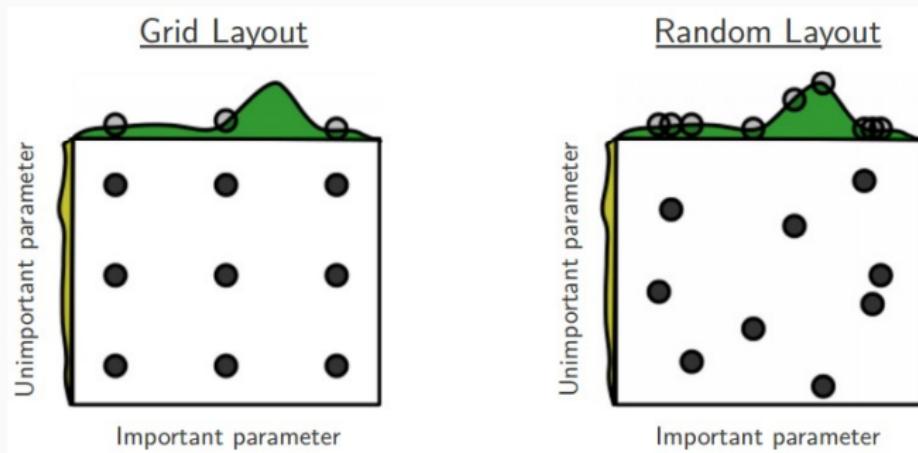
Algorithm 2 AMSGRAD

Input: $x_1 \in \mathcal{F}$, step size $\{\alpha_t\}_{t=1}^T$, $\{\beta_{1t}\}_{t=1}^T$, β_2
Set $m_0 = 0$, $v_0 = 0$ and $\hat{v}_0 = 0$
for $t = 1$ **to** T **do**
 $g_t = \nabla f_t(x_t)$
 $m_t = \beta_{1t}m_{t-1} + (1 - \beta_{1t})g_t$
 $v_t = \beta_2v_{t-1} + (1 - \beta_2)g_t^2$
 $\hat{v}_t = \max(\hat{v}_{t-1}, v_t)$ and $\hat{V}_t = \text{diag}(\hat{v}_t)$
 $x_{t+1} = \Pi_{\mathcal{F}, \sqrt{\hat{V}_t}}(x_t - \alpha_t m_t / \sqrt{\hat{v}_t})$
end for

- Но это тоже совсем не подтверждается на практике...

Практические замечания

- Ещё практические замечания об оптимизации гиперпараметров:
 - одного валидационного множества достаточно;
 - лучше гиперпараметры искать на логарифмической шкале;
 - и лучше случайным поиском, а не по сетке (Bergstra and Bengio).



Спасибо!

Спасибо за внимание!