DEEP LEARNING III: УЛУЧШАЕМ ОБУЧЕНИЕ ДАЛЬШЕ

Сергей Николенко

НИУ ВШЭ — Санкт-Петербург
27 октября 2017 г.

Random facts:

- 27 октября 312 г. Константину Великому явился Крест Господень
- 27 октября 1553 г. в Женеве сожгли Мигеля Сервета
- 27 октября 1645 г. инфант Теодозью де Браганса стал первым в истории Принцем Бразилии
- 27 октября 1962 г. --- «Чёрная суббота», день, когда мир был ближе всего к глобальной ядерной войне; в этот день американский разведывательный самолёт был сбит над Кубой советской зенитной ракетой
- 27 октября 1982 г. КНР объявила, что население Китая превысило миллиард

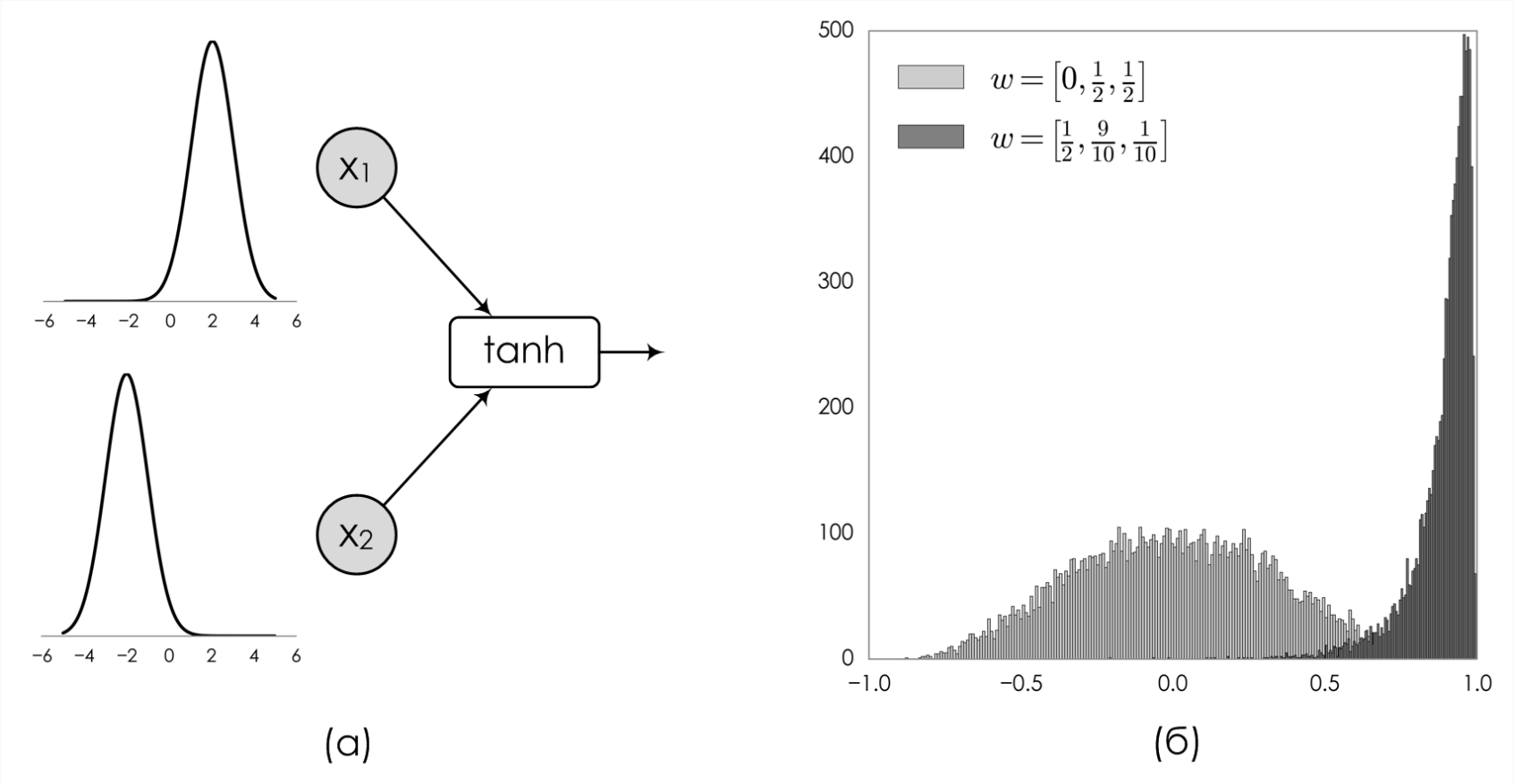
Нормализация по мини-батчам

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:

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

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

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

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

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

- 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

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

  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):

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

- 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 $B = \{\mathbf{x}_1, \dots, \mathbf{x}_m\}$:
  - вычислить базовые статистики по мини-батчу

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

  - нормализовать входы

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

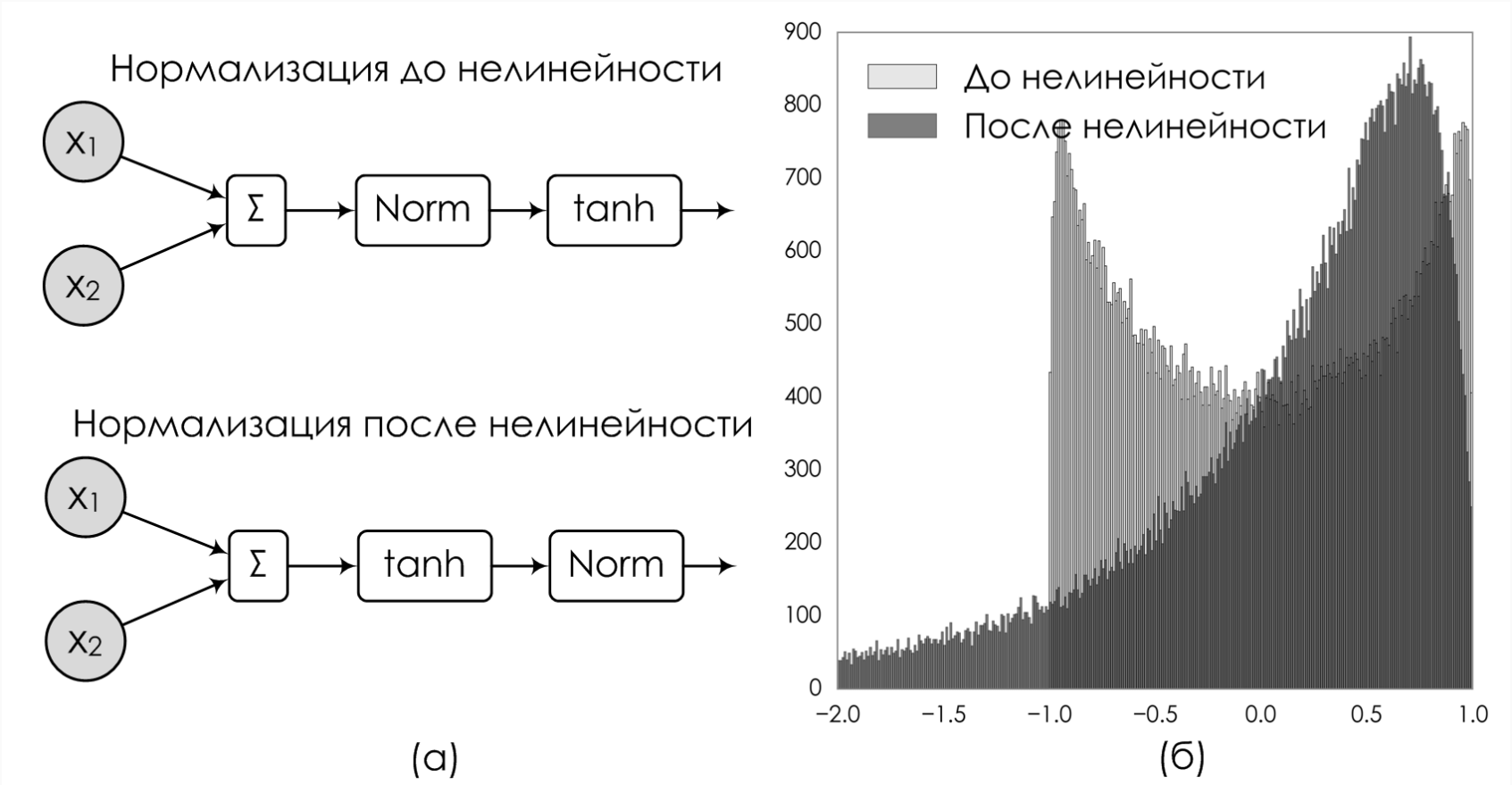
  - вычислить результат

$$\mathbf{y}_i = \gamma\mathbf{x}_i + \beta.$$

- 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.



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

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

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

Варианты градиентного спуска

МЕТОД МОМЕНТОВ

- «Ванильный» стохастический градиентный спуск:

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

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

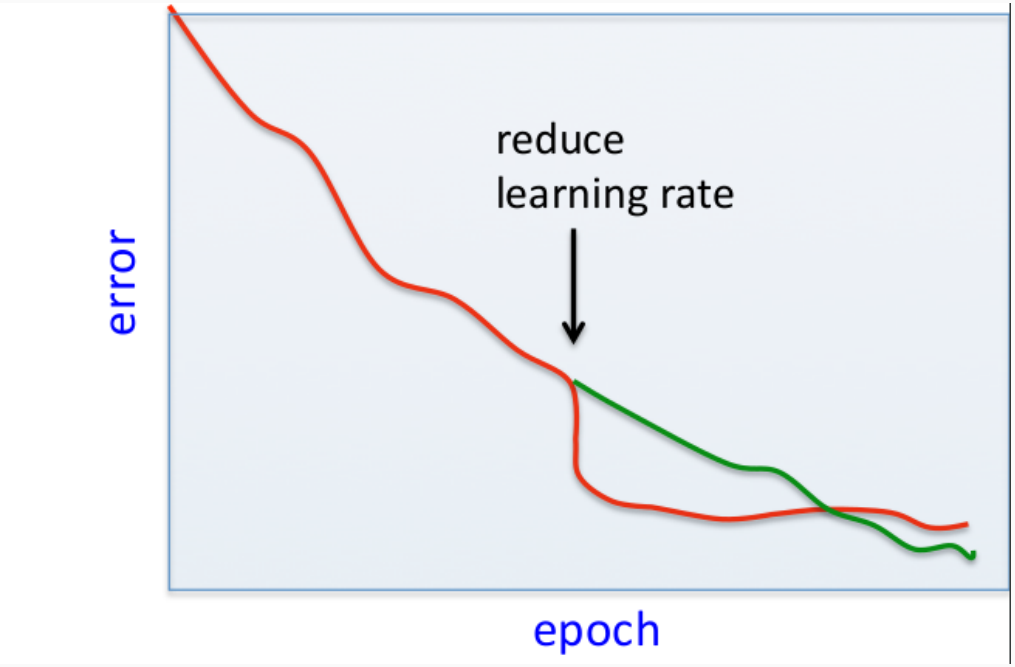
$$\eta = \eta_0 \left(1 - \frac{t}{T}\right);$$

  - или экспоненциально (exponential decay):

$$\eta = \eta_0 e^{-\frac{t}{T}}.$$

МЕТОД МОМЕНТОВ

- Скорость обучения лучше не уменьшать слишком быстро.



- Но это в любом случае никак не учитывает собственно E; лучше быть *адаптивным*.

МЕТОД МОМЕНТОВ

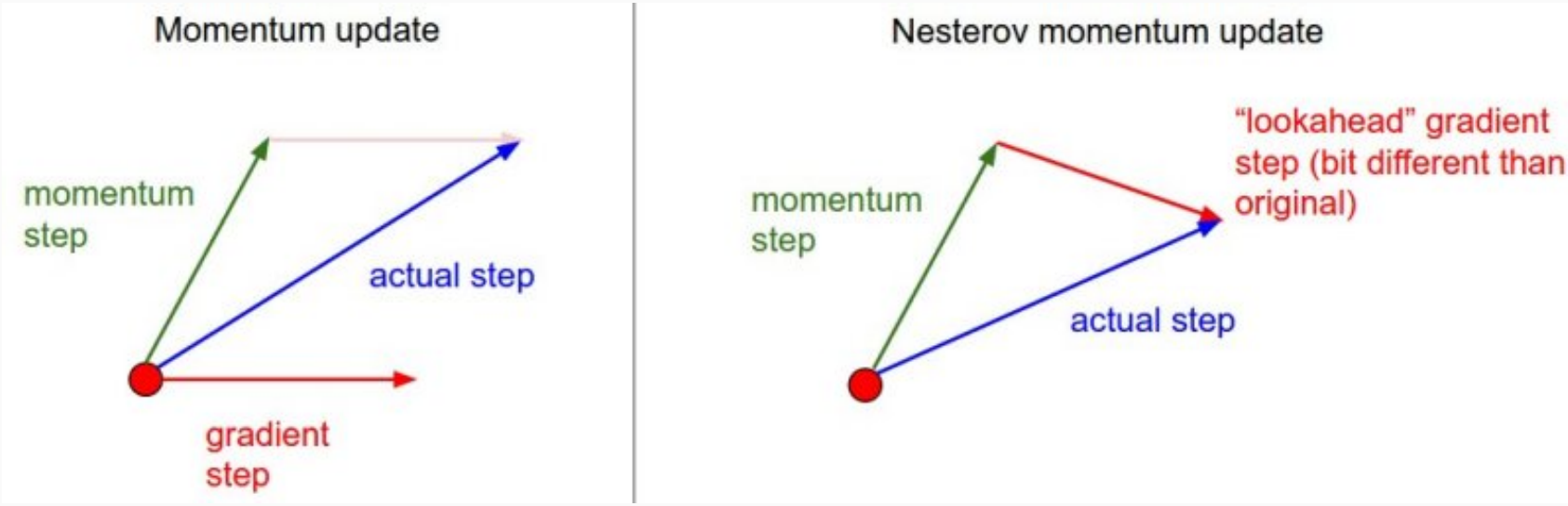
- *Метод моментов* (momentum): сохраним часть скорости, как у материальной точки.
- С инерцией получается

$$u_t = \gamma u_{t-1} + \eta \nabla_\theta E(\theta),$$
$$\theta = \theta - u_t.$$

- И теперь мы сохраняем γu_{t-1}.

МЕТОД МОМЕНТОВ

- Но на самом деле мы 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):

$$u_t = \gamma u_{t-1} + \eta \nabla_\theta E(\theta - \gamma u_{t-1})$$

- Можно ли ещё лучше?..

МЕТОД МОМЕНТОВ

- ...ну, можно попробовать методы второго порядка.
- Метод Ньютона:

$$E(\theta) \approx E(\theta_0) + \nabla_\theta E(\theta_0)(\theta - \theta_0) + \frac{1}{2}(\theta - \theta_0)^\top H(E(\theta))(\theta - \theta_0).$$

- Когда работает, это обычно гораздо быстрее, и нет никакой η, ничего настраивать не надо.
- Но нужно считать гессиан $H(E(\theta))$, и это нереально.

МЕТОД МОМЕНТОВ

- Есть, правда, приближения.
- 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 лоб» было бы нужно считать градиент по всему датасету, а по мини-батчам непонятно как).

Адаптивные методы градиентного спуска

- Но дальше улучшить всё равно можно.
- Заметим, что до сих пор скорость обучения была одна во всех направлениях.
- 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 $g_{t,i} = \nabla_{\theta_i} L(\theta)$, получим

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

  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 θ_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:

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

- 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 $\Delta\theta$ — 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 $\Delta\theta$ зависели от отношений градиентов, и величина обновлений вовсе безразмерная.

АДАПТИВНЫЕ МЕТОДЫ ГРАДИЕНТНОГО СПУСКА

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

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

- 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:

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

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

- 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

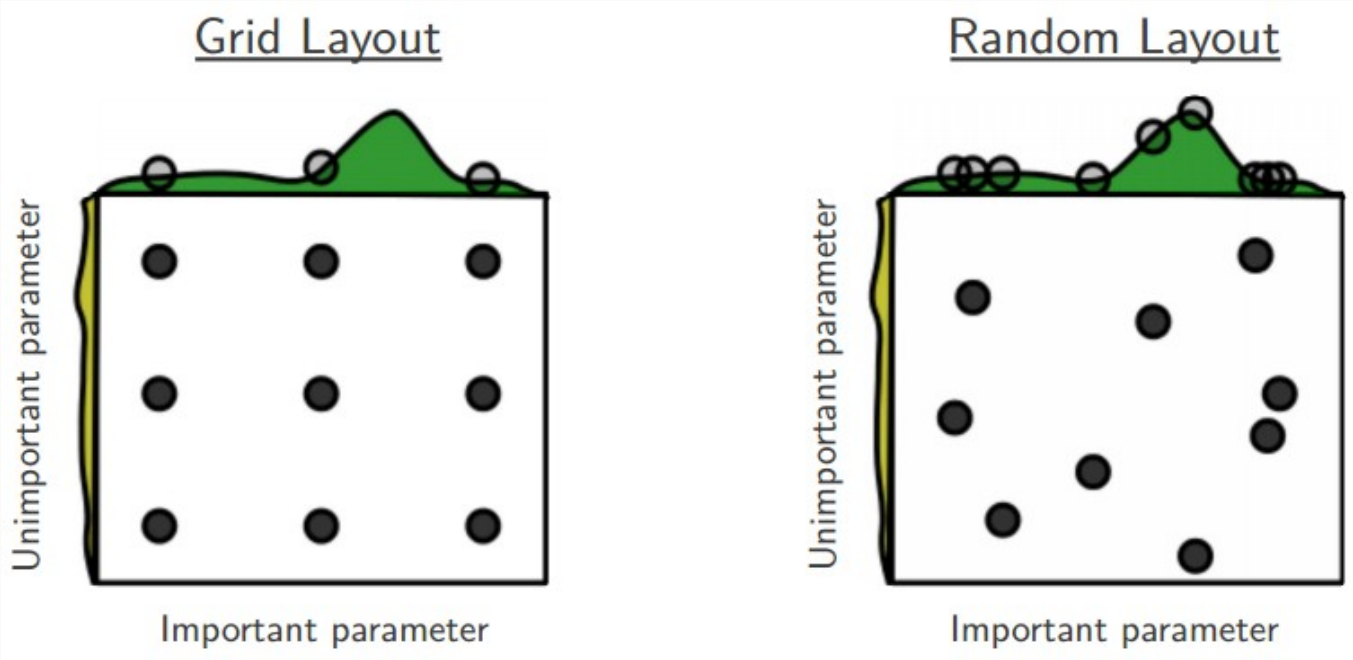
- И последний алгоритм – *Adam* (Kingma, Ba, 2014).
- Модификация *Adagrad* со сглаженными версиями среднего и среднеквадратичного градиентов:

$$
\begin{aligned}
m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t, \\
v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \\
u_t &= \frac{\eta}{\sqrt{v + \epsilon}} m_t.
\end{aligned}
$$

- (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.
- ...[анимированные примеры]...

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).



THANK YOU!

Thank you for your attention!