

# Сверточные нейронные сети

---

Сергей Николенко

НИУ ВШЭ – Санкт-Петербург

26 сентября 2020 г.

---

*Random facts:*

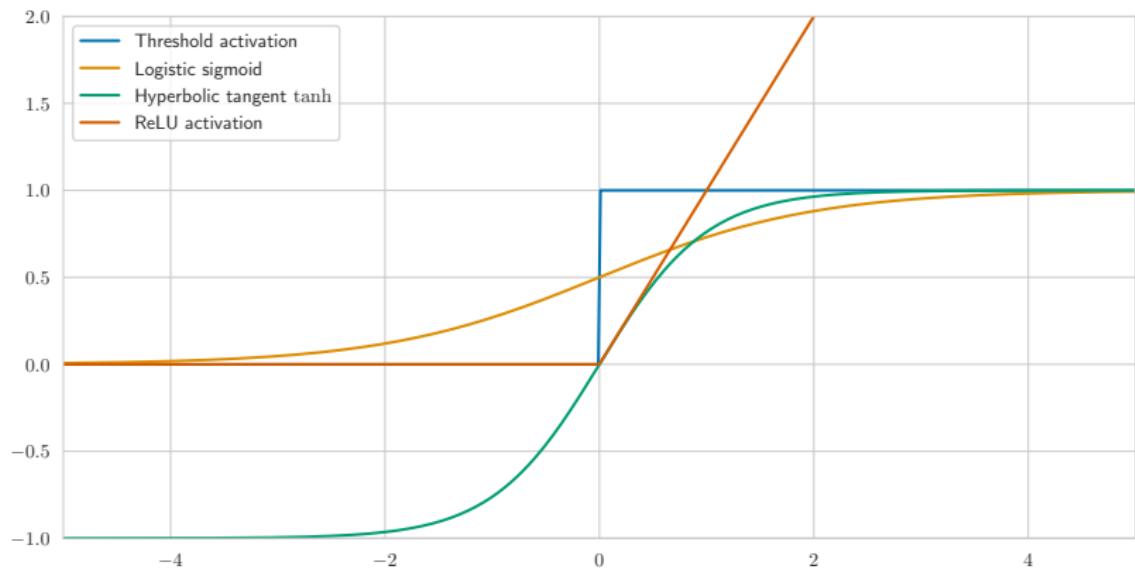
- 26 сентября — Европейский день языков и Всемирный день контрацепции
- 26 сентября 1580 г. Фрэнсис Дрейк на «Золотой лани» вернулся в Плимут из кругосветного путешествия (второго после Магеллана)
- 26 сентября 1687 г. во время обстрела Афин венецианской армией был разрушен Парфенон
- 26 сентября 1983 г. подполковник Станислав Петров предотвратил потенциальную ядерную войну, когда из-за сбоя в системе предупреждения о ракетном нападении поступило ложное сообщение об атаке со стороны США
- 26 сентября 2008 г. в Монголии открылась крупнейшая конная статуя в мире, памятник Чингисхану высотой 40 метров (плюс 10м постамента); на голове лошади расположена смотровая площадка
- 26 сентября 1934 г. СССР вошёл в состав Лиги наций; хватило лет на пять

## О функциях активации

---

# Ещё о функциях активации

- Немножко ещё о функциях активации; мы уже обсуждали логистический сигмоид  $\sigma(a) = \frac{1}{1+e^{-a}}$ , гиперболический тангенс  $\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$  и ReLU:



## Ещё о функциях активации

- Ещё мы обсуждали разные варианты ReLU:

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

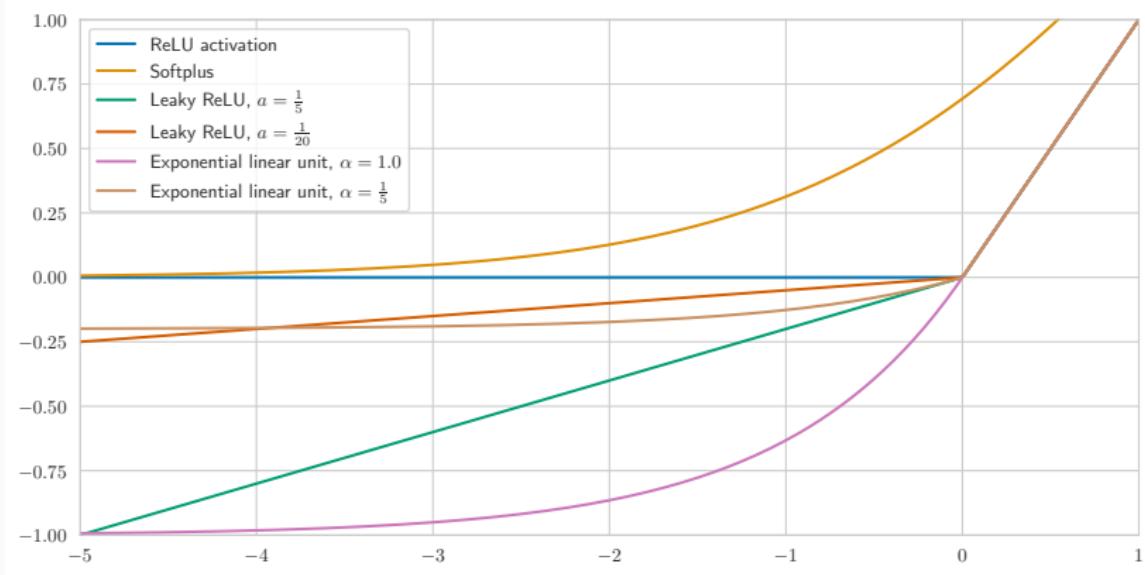
$$\cdot \text{LReLU}(x) = \begin{cases} ax, & \text{if } x < 0, \\ x, & \text{if } x > 0 \end{cases}$$

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

$$\cdot \text{softplus}(x) = \ln(1 + e^x).$$

# Ещё о функциях активации

- Ещё мы обсуждали разные варианты ReLU:

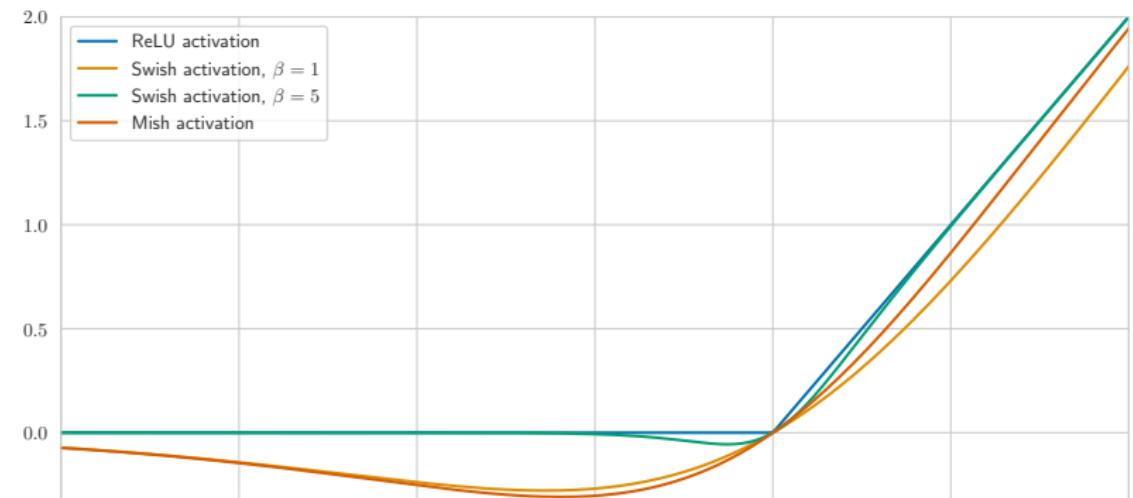


## Ещё о функциях активации

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



## Ещё о функциях активации

- Но самое интересное произошло в десятых числах сентября 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

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

## Ещё о функциях 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  $\text{ReLU}(x) = \max(x, 0)$ , а гладкий максимум

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

т.е. в точности *Swish*!

- для  $g(x) = x$  и  $h(x) = ax$  при некотором  $a < 1$ , жёсткий максимум будет  $\text{LReLU}(x) = \max(x, px)$ , а гладкий максимум

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

## Ещё о функциях активации

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

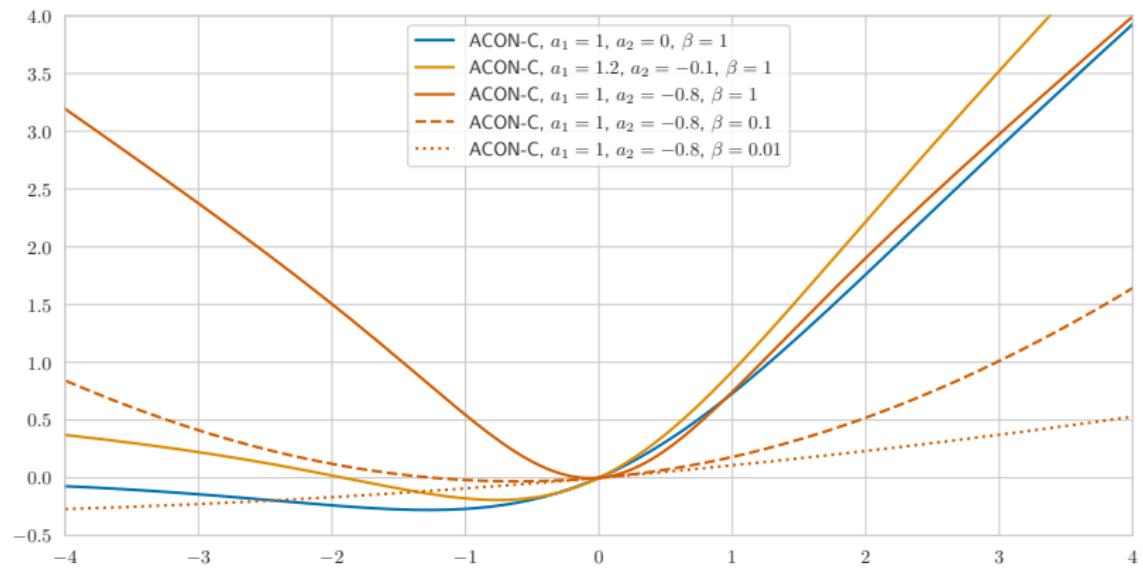
$$f_{ACON-C} = S_\beta(a_1x, a_2x) = (a_1 - a_2)x\sigma(\beta(a_1 - a_2)x) + a_2x;$$

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

$$\lim_{x \rightarrow \infty} \frac{df_{ACON-C}}{dx} = a_1, \quad \lim_{x \rightarrow -\infty} \frac{df_{ACON-C}}{dx} = a_2.$$

# Ещё о функциях активации

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



# Ещё о функциях активации

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

Table 4: **Comparison with other activations.** We report the top-1 error on the ImageNet dataset.

Activation	Top-1 err.
ReLU	39.4
Swish [40]	38.3
Mish [37]	39.5
ELU [7]	39.5
SoftPlus[10]	39.6
ACON-C	37.0
meta-ACON	<b>34.8</b>

Table 6: **Meta-ACON v.s. SENet** [19]. We report the ImageNet top-1 error rates of ShuffleNetV2 and ResNet.

	Baseline	SE	meta-ACON
ShuffleNetV2 0.5x	39.4	37.5	<b>34.8</b>
ShuffleNetV2 1.5x	27.4	26.4	<b>24.7</b>
ResNet-50	24.0	22.8	<b>22.0</b>
ResNet-101	22.8	21.9	<b>21.1</b>
ResNet-152	22.3	21.5	<b>20.5</b>

Table 5: **Design space in meta-ACON.** We report the top-1 error on the ImageNet dataset. Comparison on 3 different levels of design space. We give 3 most simple examples on ShuffleNetV2 0.5x.  $fc$  denotes fully-connected,  $\sigma$  denotes sigmoid,  $GAP$  denotes global average pooling.

	manner	Top-1 err.
baseline	-	39.4
pixel-wise	$\sigma(x)$	37.2
channel-wise	$\sigma(fc[fc[GAP(x)]]))$	34.8
layer-wise	$\sigma[\sum_c GAP(x)]$	36.3

Table 7: **Comparison on the extremely deep SENet-154** [19]. We report the ImageNet top-1 error rates. We implement all the models by ourselves.

Activation	Top-1 err.
ReLU	18.95
ACON-A (Swish)	19.02
ACON-C	<b>18.40</b>

- К чему всё это приведёт — пока бог знает...

## 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  $\alpha$ .
- Первая мысль — пусть  $\alpha$  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  $\alpha$  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

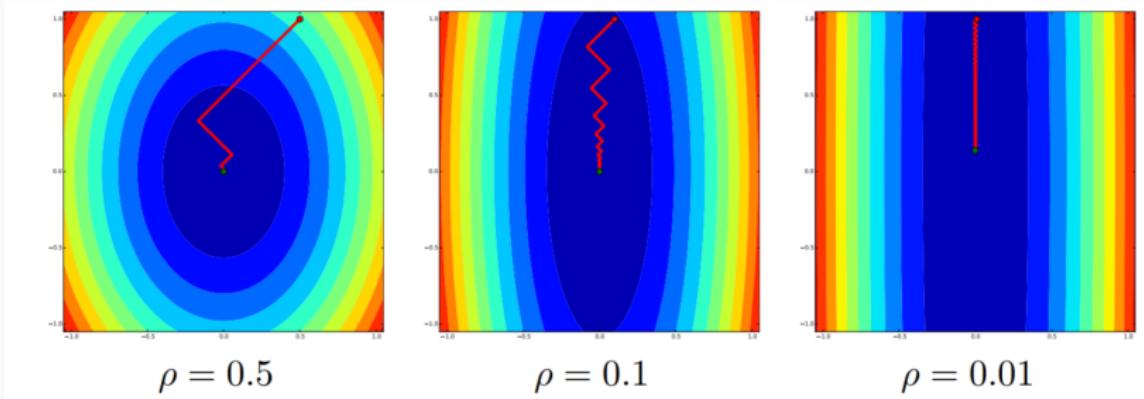
$$|\phi'_k(\alpha)| \leq c_2 |\phi'_k(0)|,$$

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

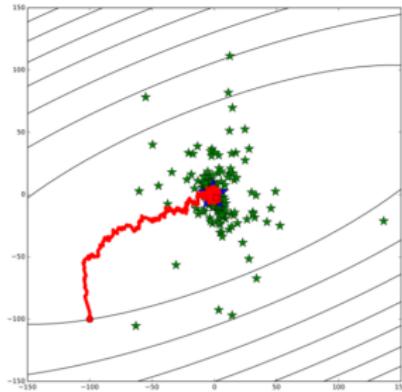
- Да просто эмпирические оценки общей функции по подвыборке:

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

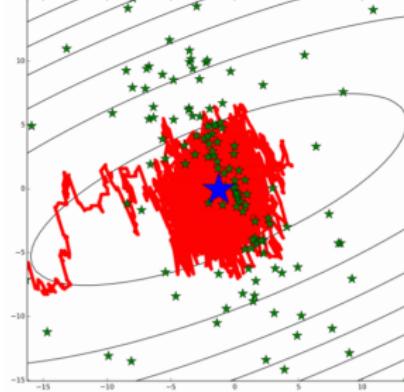
- Это очень хорошие оценки: несмешённые, сходятся на бесконечности (правда, медленно), легко посчитать.
- В целом, так и мотивируется стохастический градиентный спуск (SGD): метод Монте-Карло по сути.
- Но есть проблемы...

# 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:

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

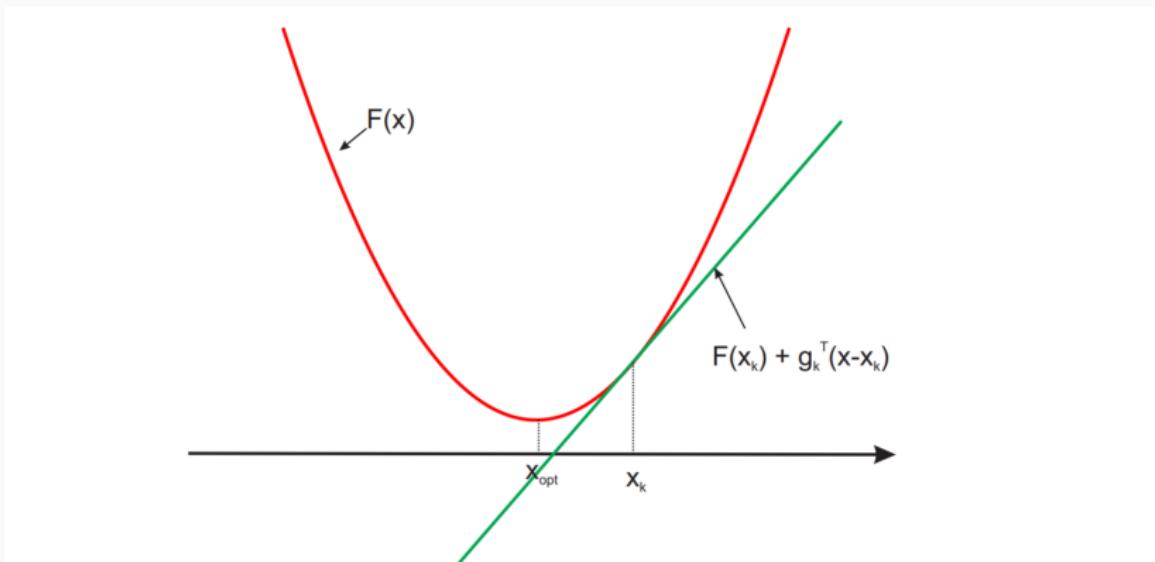
- Возьмём ожидание по  $q(y)$  в момент времени  $k$ :

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

# 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  $\alpha_i$ . 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  $\alpha_i = h$

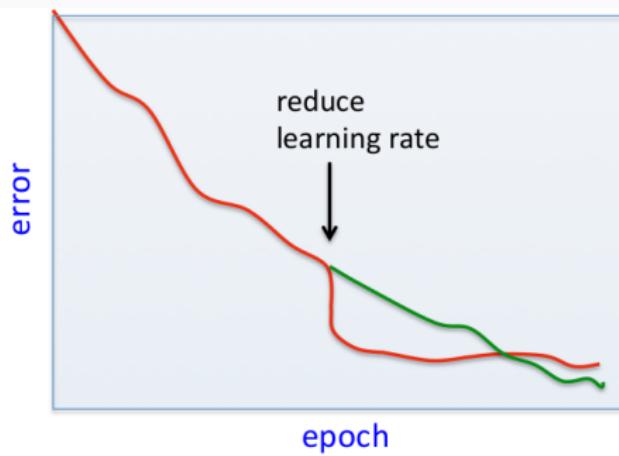
$$\mathbb{E}F(\hat{\mathbf{x}}_k) - F(\mathbf{x}_{\text{opt}}) \leq \frac{R^2}{2h(k+1)} + \frac{G^2h}{2} \xrightarrow{k \rightarrow \infty} \frac{G^2h}{2}.$$

# 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  $\gamma u_{t-1}$ .



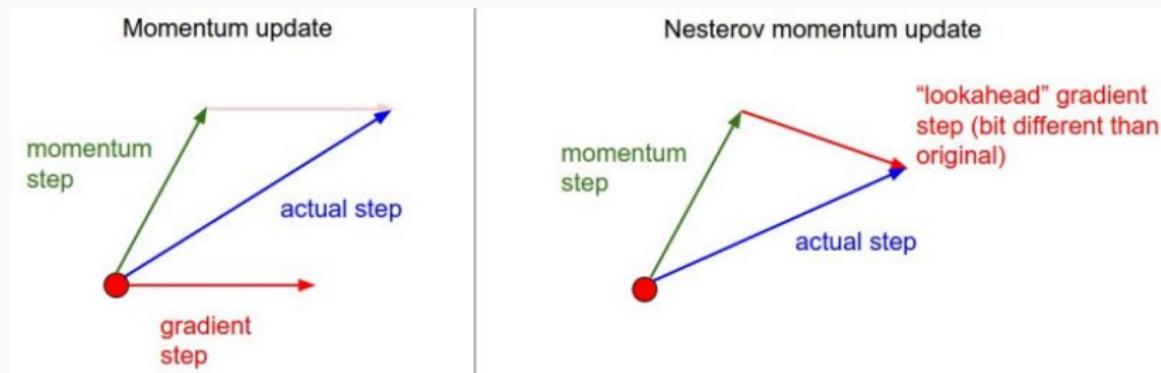
without momentum



with momentum

# Метод моментов

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



# Метод моментов

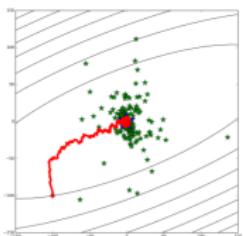
- *Метод Нестерова* (Nesterov's momentum):

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

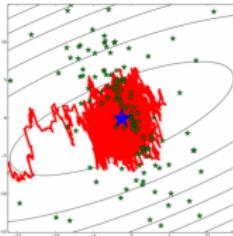


# Метод моментов

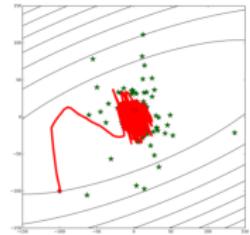
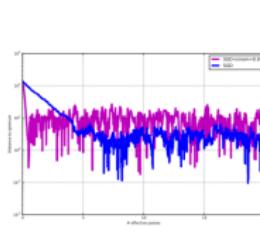
- Всё равно, конечно, проблемы не пропадают:



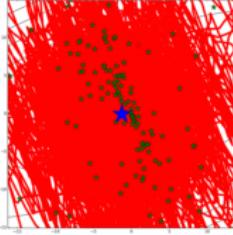
SGD



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

## Адаптивные методы градиентного спуска

- Эта проблема решается в методе второго порядка:  
обновление параметров  $\Delta w$  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:

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

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

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

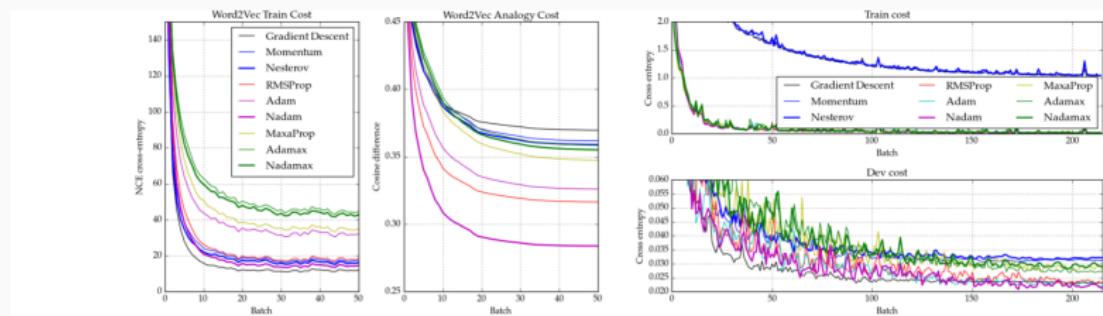
- (Kingma, Ba, 2014) рекомендуют  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$ ,  $\epsilon = 10^{-8}$ .
- *Adam* практически не требует настройки, используется на практике очень часто.
- ...[<http://ruder.io/optimizing-gradient-descent/>]

# 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:

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

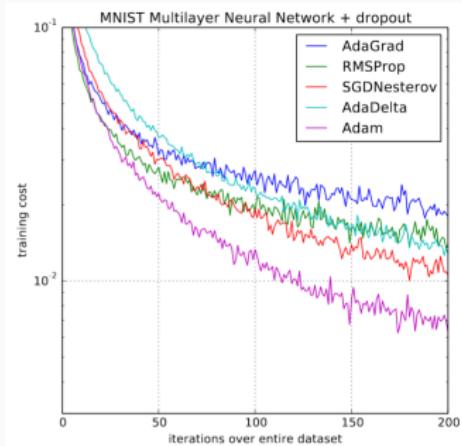
где обычно  $H_k = \text{diag} \left( \left[ \sum_{i=1}^k \eta_i g_i \circ g_i \right]^{1/2} \right)$ , где

$g_k = \tilde{\nabla}f(w_k + \gamma_k (w_k - w_{k-1}))$  (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:

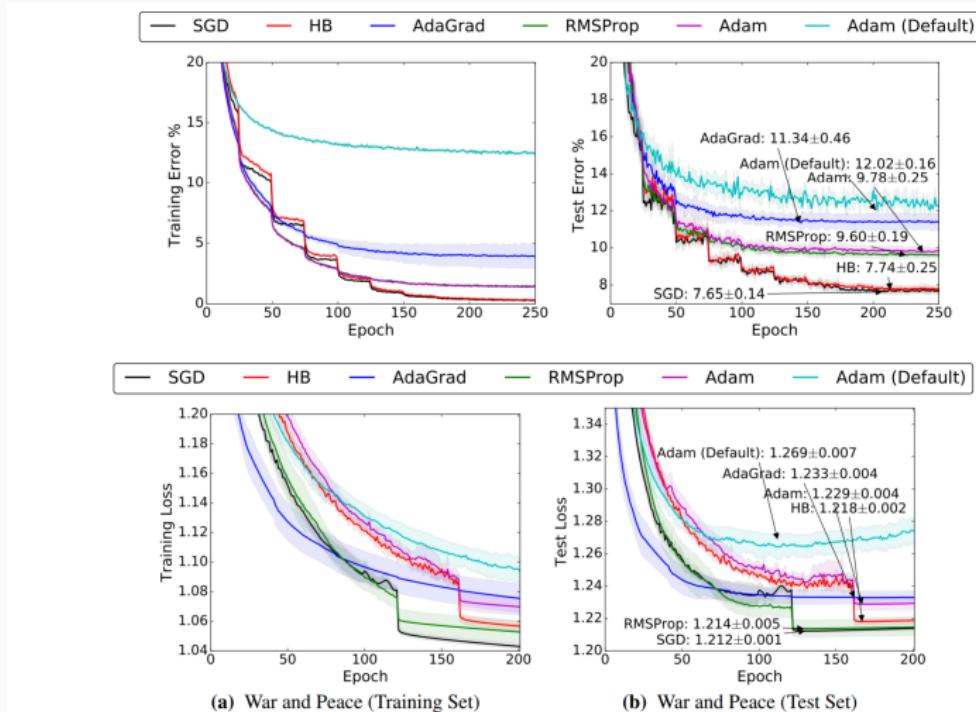


- Но потом выяснилось, что не всё так просто... часто применяли простой SGD. Почему?

- 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.*

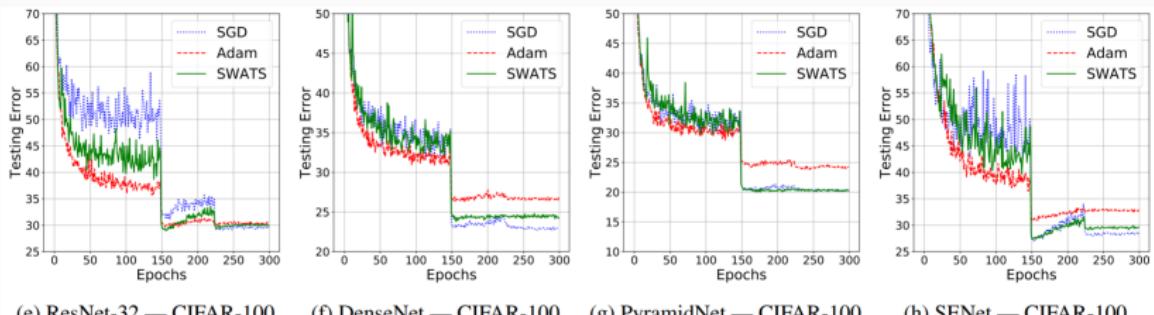
# Adam, AdamW и другие животные

- 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)



- 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,  $\eta$  – 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 L<sub>2</sub> 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 / L<sub>2</sub> 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  $\alpha$  и  $w$  завязан друг на друга.
- Решение – просто перенести decay в само изменение  $\mathbf{x}_t$  (строка 9) и добавить масштабирование  $\eta_t$ .

# Adam, AdamW и другие животные

- То же самое происходит и с Adam:

---

**Algorithm 2** Adam with L<sub>2</sub> regularization and  
Adam with weight decay (AdamW)

---

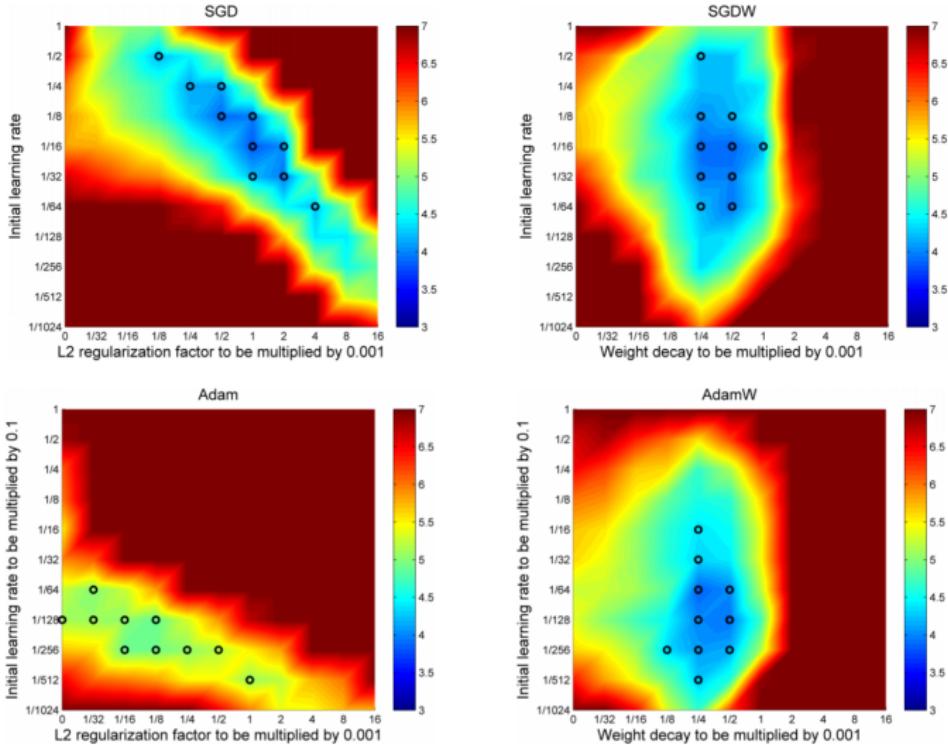
```
1: given  $\alpha = 0.001$ ,  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$ ,  $\epsilon = 10^{-8}$ ,  $w \in \mathbb{R}$ 
2: initialize time step  $t \leftarrow 0$ , parameter vector  $\mathbf{x}_{t=0} \in \mathbb{R}^n$ , first
   moment vector  $\mathbf{m}_{t=0} \leftarrow \mathbf{0}$ , second moment vector  $\mathbf{v}_{t=0} \leftarrow \mathbf{0}$ ,
   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:   $\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \eta_t \left( \alpha \hat{\mathbf{m}}_t / (\sqrt{\hat{\mathbf{v}}_t} + \epsilon) + w\mathbf{x}_{t-1} \right)$ 
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.

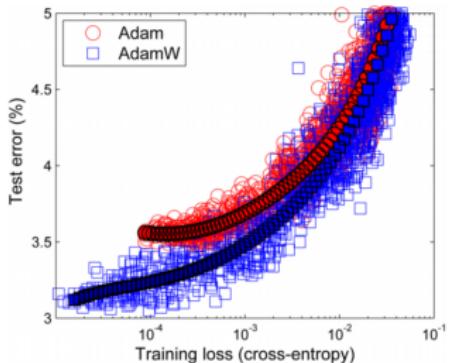
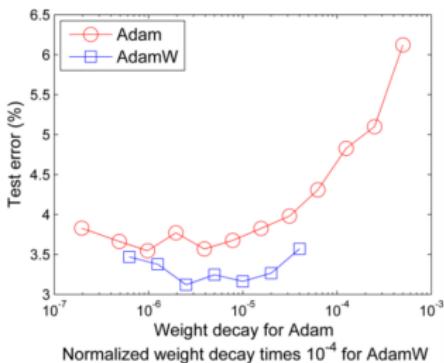
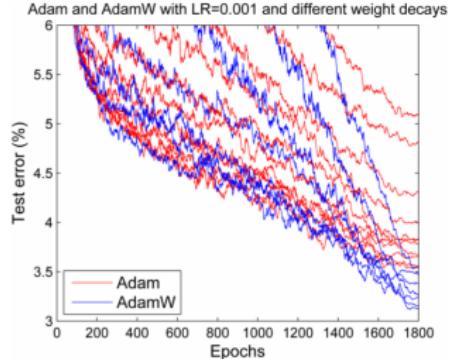
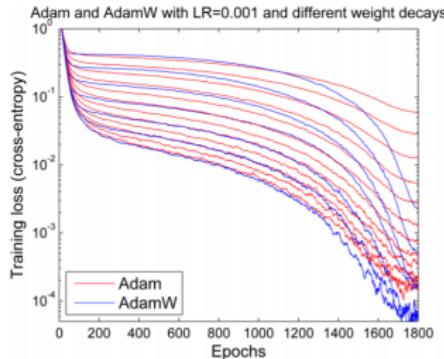
# Adam, AdamW и другие животные

- И теперь гиперпараметры разделяются лучше:



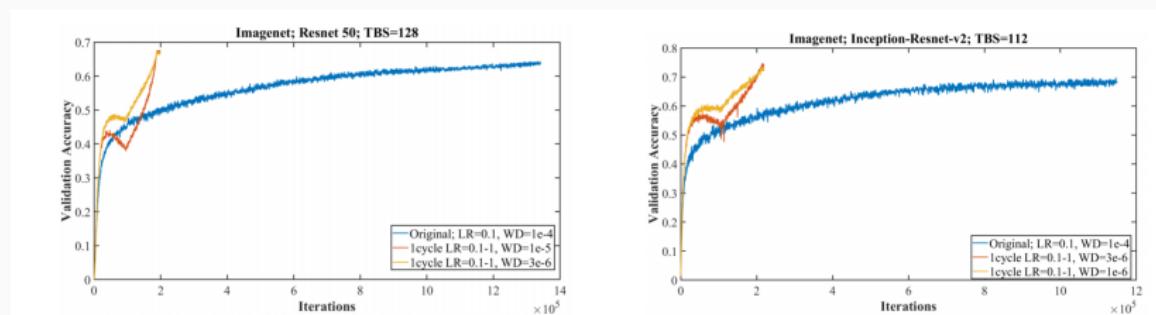
# Adam, AdamW и другие животные

- 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

---

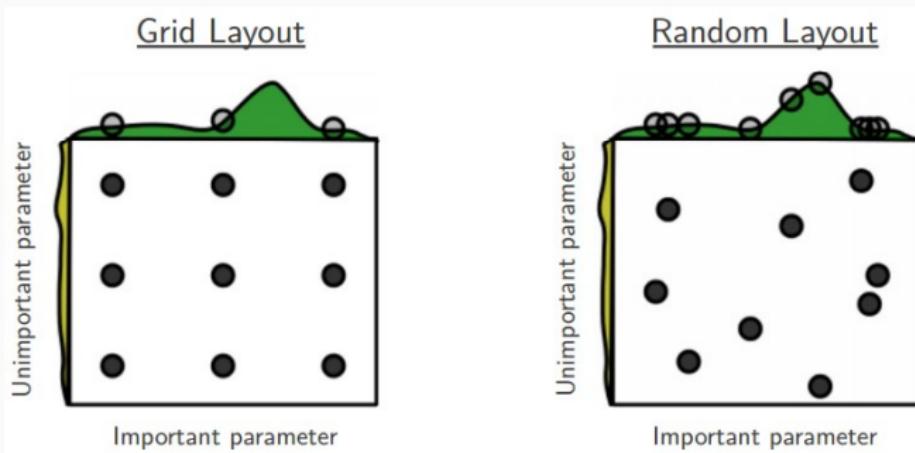
```
Input:  $x_1 \in \mathcal{F}$ , step size  $\{\alpha_t\}_{t=1}^T$ ,  $\{\beta_{1t}\}_{t=1}^T$ ,  $\beta_2$ 
Set  $m_0 = 0$ ,  $v_0 = 0$  and  $\hat{v}_0 = 0$ 
for  $t = 1$  to  $T$  do
     $g_t = \nabla f_t(x_t)$ 
     $m_t = \beta_{1t}m_{t-1} + (1 - \beta_{1t})g_t$ 
     $v_t = \beta_2v_{t-1} +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



## Свёрточные сети: идея

---

# Зрительная кора

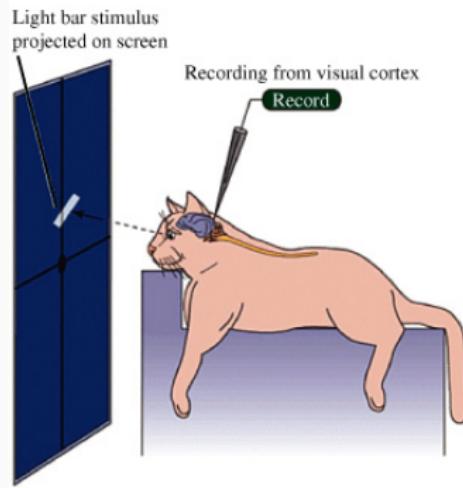
- Зрительная кора давно известна.
- Но до середины XX века изучать могли только на очень «высоком» уровне: болезни, травмы, агнозии.
- Хьюбел и Визель: исследования отдельных нейронов, на что они активируются.



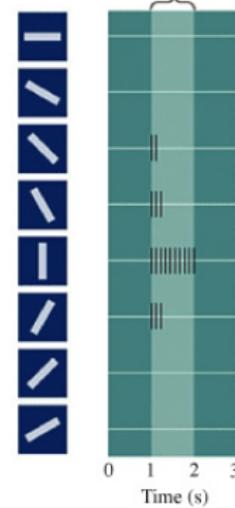
# Зрительная кора

- Оказалось, что нейроны «ближних» к сетчатке уровней активируются на простые формы.
- Вообще оказалось, что зрительная кора делится на уровни: V1, V2, V3...
- Но от V3 и выше Хьюбел и Визель уже мудро не стали исследовать.

A Experimental setup

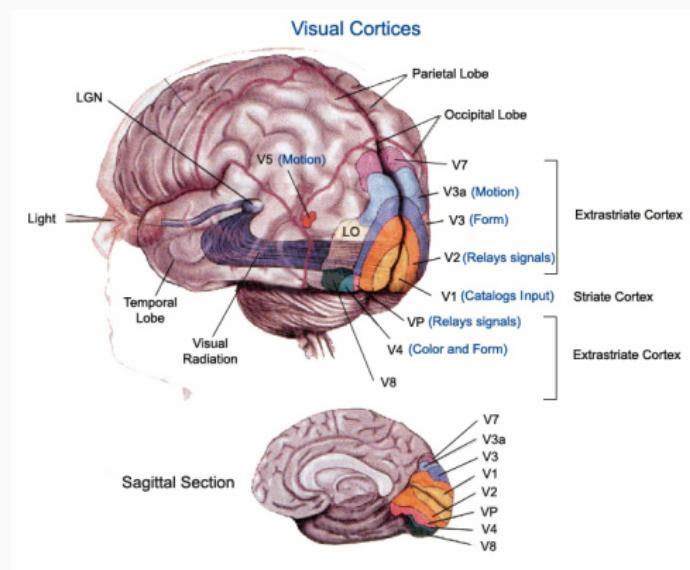


B Stimulus orientation



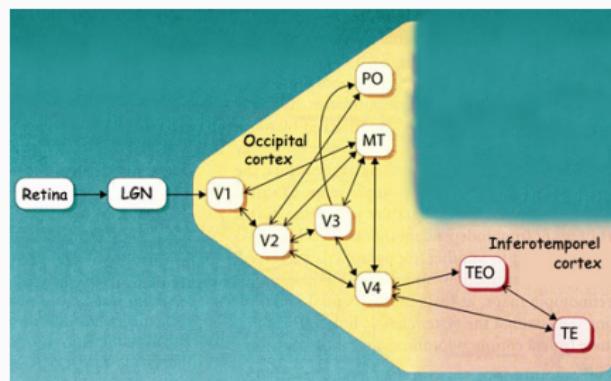
# Зрительная кора

- Сетчатка → латеральное коленчатое тело (LGN, lateral geniculate nucleus) → зрительная кора.
- И там есть чёткая структура, организованная в уровни от V1 до V4-V6.



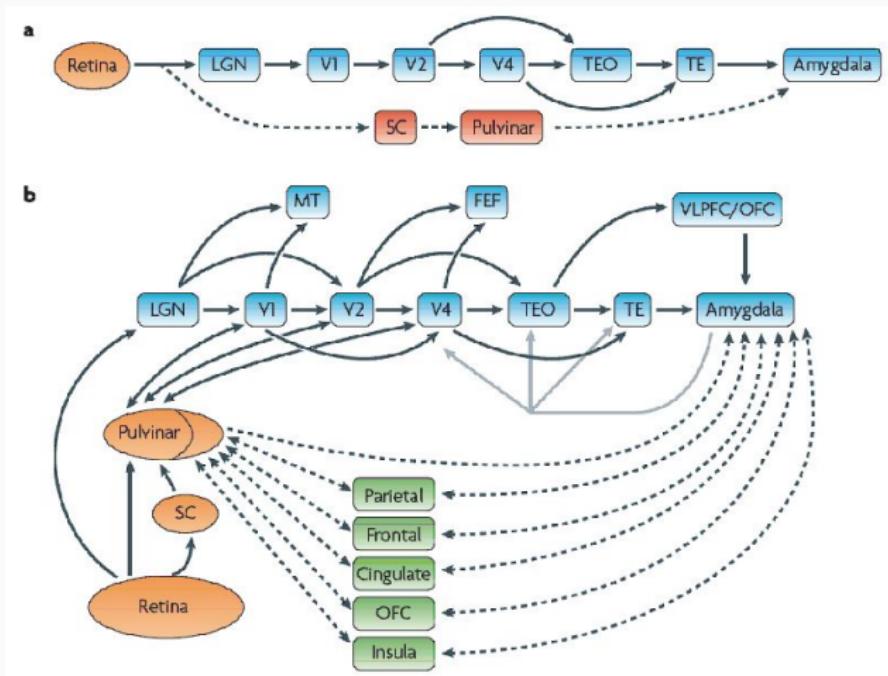
# Зрительная кора

- Ну, вроде того. Мозг – сложная штука.



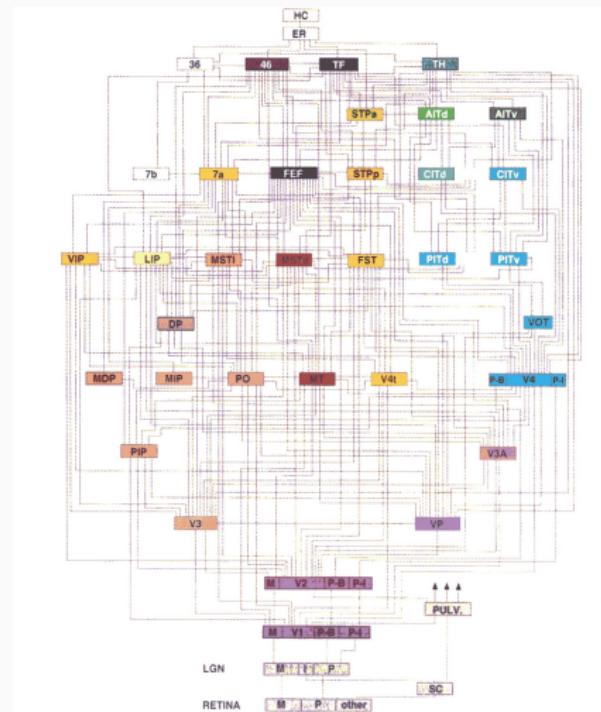
# Зрительная кора

- Правда сложная.



# Зрительная кора

- Честно-честно.

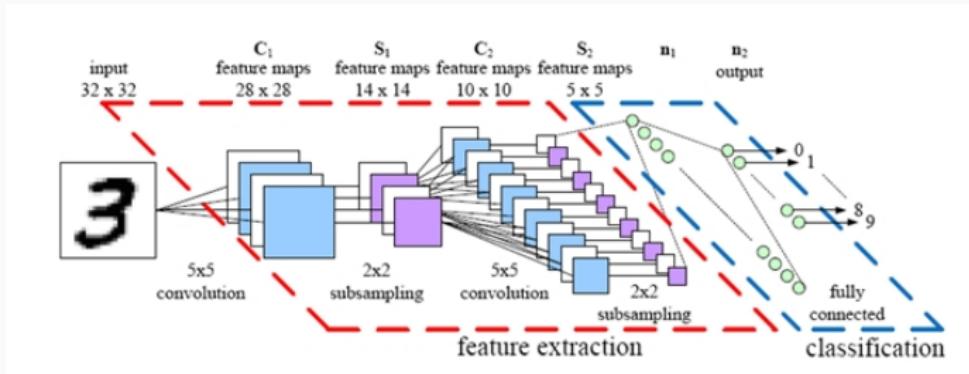


# Зрительная кора

- Но некоторые условные уровни там действительно есть:
  - V1 – локальные признаки, топографические карты;
  - V2 – больше локальных признаков, бинокулярное зрение;
  - V3 – цвет, текстура, первые результаты сегментации;
  - V4 – геометрические формы, силуэты; это самый важный уровень для *внимания*;
  - V5 – распознаёт движения объектов, сегментированных на V4;
  - V6 – обобщает данные со всей картинки, wide-field stimulation; например, обрабатывает наши собственные движения;
  - V7 (наверное) – распознаёт сложные объекты, например человеческие лица.
- What pathway: V2 – V4, распознавание форм и объектов.
- Where pathway: V2 – V5 – V6, распознавание движений, управление глазами и руками.

# Свёрточные нейронные сети: идея

- Эти идеи нашли отражение в искусственных сетях.
- Свёртки впервые появились в *Neocognitron* (Fukushima, 1979; 1980).
- В современной форме – в группе Лекуна во второй половине 1980-х.
- Главная идея: хочется применить одну и ту же операцию к разным частям изображения.



# Свёрточные нейронные сети: идея

- Разобьём изображение на окна:



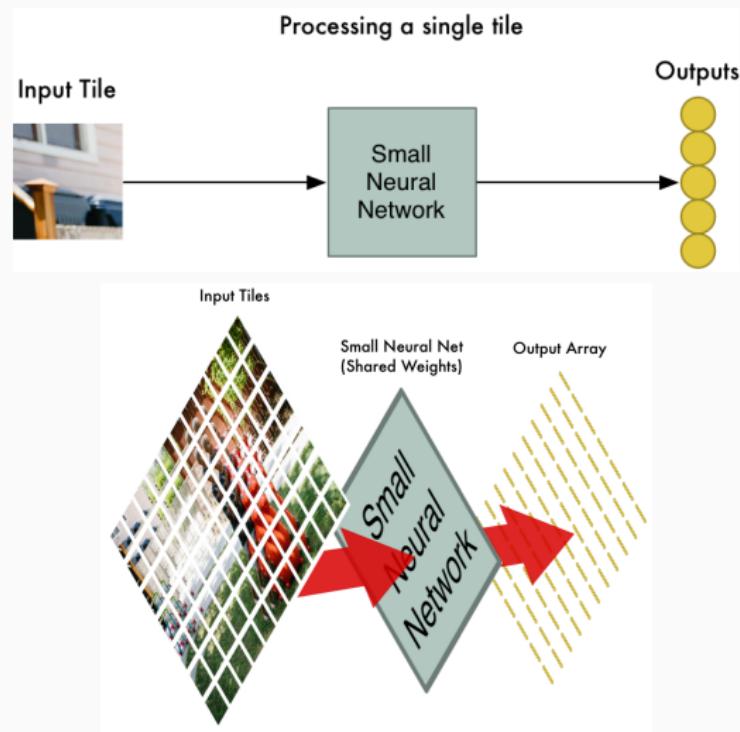
# Свёрточные нейронные сети: идея

- Разобьём изображение на окна:



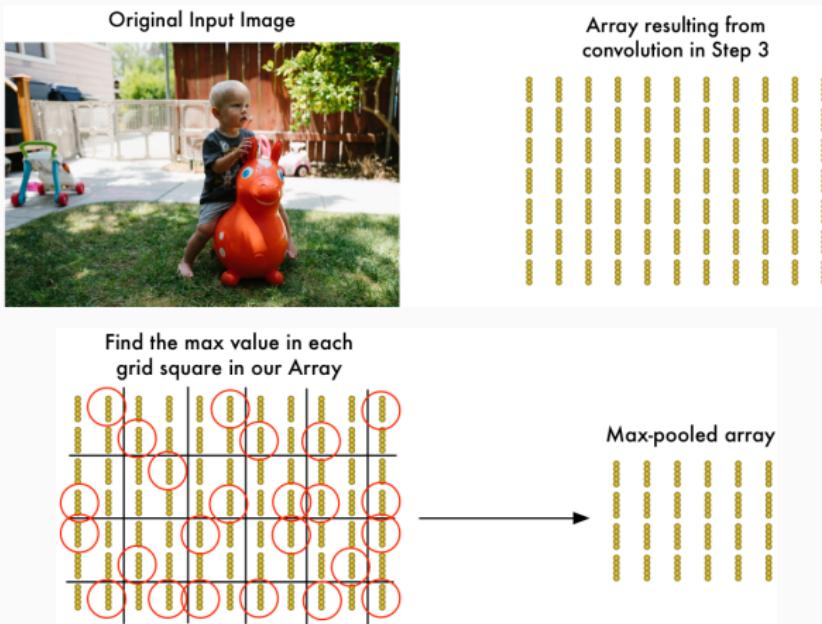
# Свёрточные нейронные сети: идея

- Применим маленькую нейронную сеть к каждому окну:



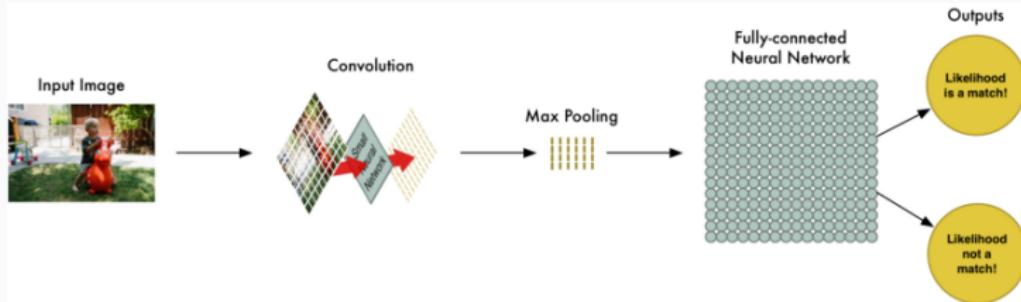
# Свёрточные нейронные сети: идея

- Затем сожмём результат, сделав субдискретизацию (pooling; например, max-pooling) по маленьким окнам:

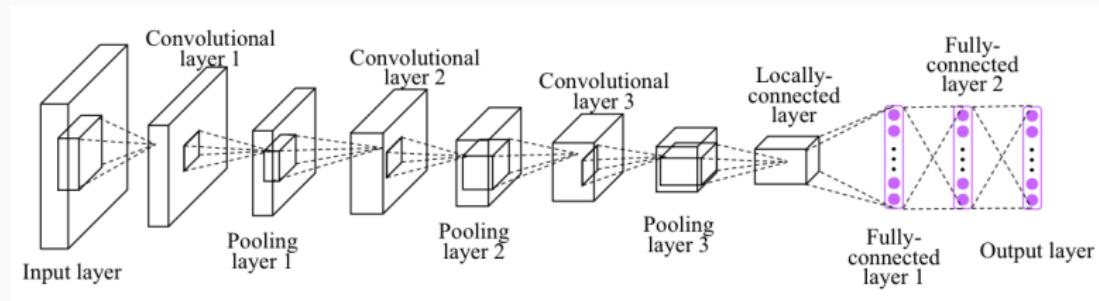


# Свёрточные нейронные сети: идея

- А затем уже используем эти признаки, чтобы делать предсказания, например, полно связной сетью:

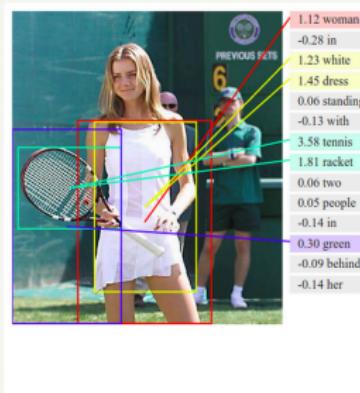
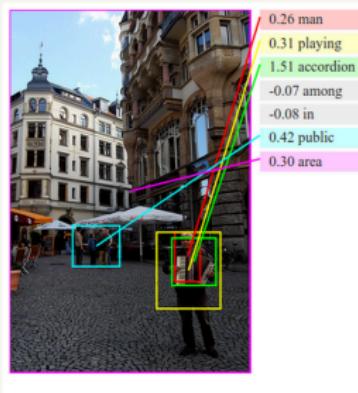
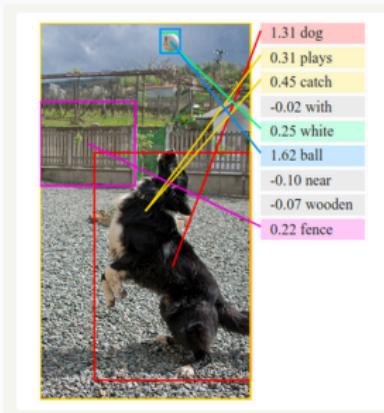


- Добавляя новые уровни, можно улучшить обобщающую способность, сделать признаки более общими:



# Свёрточные нейронные сети: идея

- Более того, можно посмотреть, какие именно части картинки вызывают активации отдельных нейронов:



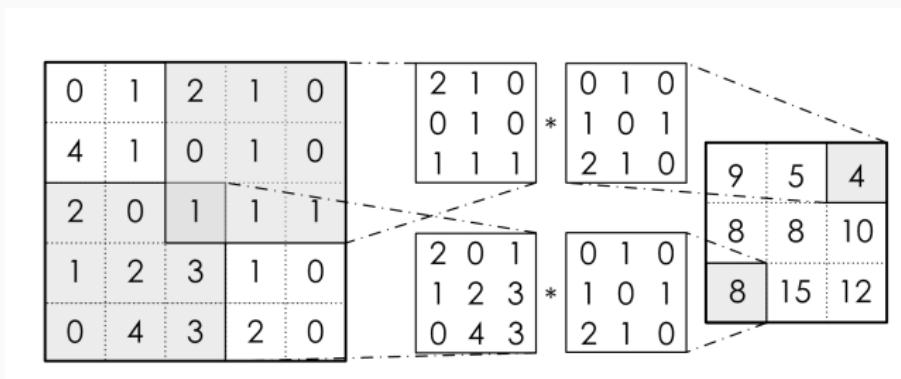
## Свёрточные нейронные сети: идея

- Основная идея: мы применяем *одни и те же веса* по всей картинке.
- Это автоматически даёт устойчивость к переносам и т.п.
- И радикально сокращает число весов.
- Т.е. свёрточная структура – это такая радикальная форма регуляризации.

# CNN формально

- Формально говоря, мы к каждому окну применяем одну и ту же маленькую матрицу.
- Если  $x^l$  – карта признаков на слое  $l$ , то двумерная свёртка размера  $2d + 1$  с матрицей весов  $W$  размера  $(2d + 1) \times (2d + 1)$  выглядит как

$$y_{i,j}^l = \sum_{-d \leq a,b \leq d} W_{a,b} x_{i+a, j+b}^l.$$



## CNN формально

- Если  $x^l$  – карта признаков на слое  $l$ , то двумерная свёртка размера  $2d + 1$  с матрицей весов  $W$  размера  $(2d + 1) \times (2d + 1)$  выглядит как

$$y_{i,j}^l = \sum_{-d \leq a,b \leq d} W_{a,b} x_{i+a, j+b}^l.$$

- Затем обычно идёт нелинейность:

$$z_{i,j}^l = h(y_{i,j}^l).$$

- В современных сетях – почти всегда ReLU.

# CNN формально

- И затем max-pooling (иногда average-pooling, но обычно max):

$$x_{i,j}^{l+1} = \max_{-d \leq a,b \leq d} z_{i+a,j+b}^l.$$

0	1	2	1
4	1	0	1
2	0	1	1
1	2	3	1

(a)

4	2	2
4	1	1
2	3	3

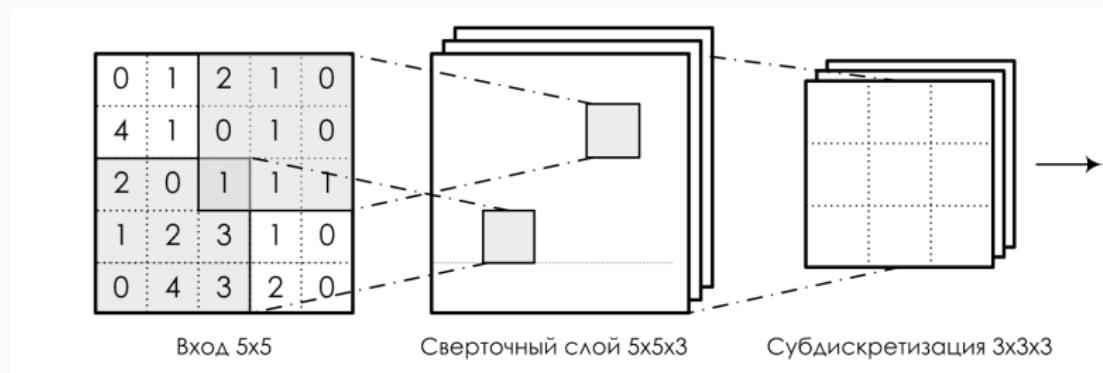
(б)

4	2
2	3

(в)

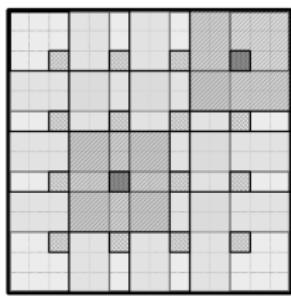
# CNN формально

- Вот и получилась общая схема одного свёрточного слоя:

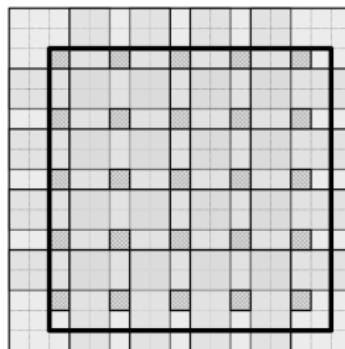


# CNN формально

- Окнами можно покрывать по-разному; обычно всё-таки сохраняют исходный размер:



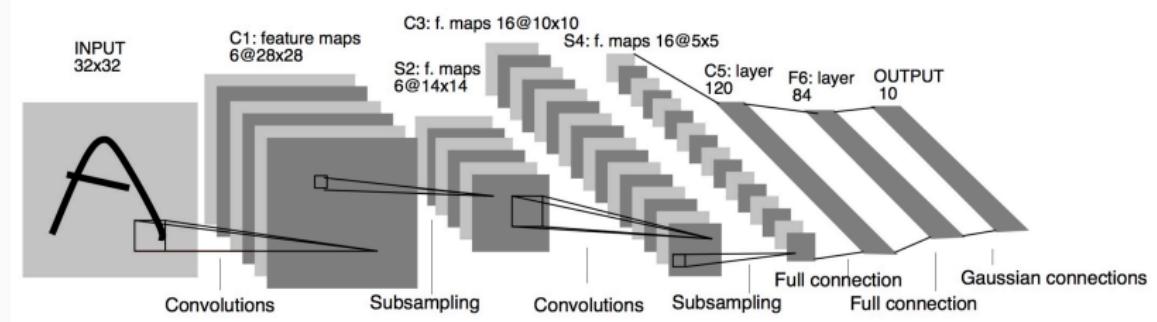
(а) окна 5x5 с шагом в 3 пикселя,  
`padding='VALID'`



(б) окна 5x5 с шагом в 3 пикселя,  
`padding='SAME'`

# CNN формально

- Классическая архитектура *LeNet*:



- Современные CNN стали очень глубокими за счёт новых трюков – к ним мы скоро перейдём.

Спасибо!

---

Спасибо за внимание!