

# Оптимизация в задачах машинного обучения

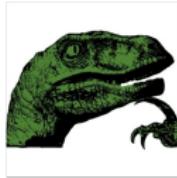
Алексей Романов

[alexey.romanov@phystech.edu](mailto:alexey.romanov@phystech.edu)

МФТИ, ABBYY

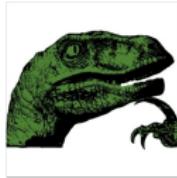
06.10.2017

# Вопросы для разминки



Что такое машинное обучение?

# Вопросы для разминки



Что такое машинное обучение?

Чем алгоритмы машинного обучения отличаются друг от друга?

# Вопросы для разминки



Что такое машинное обучение?

Чем алгоритмы машинного обучения отличаются друг от друга?

Какими свойствами должна обладать функция потерь?

# Вопросы для разминки



Что такое машинное обучение?

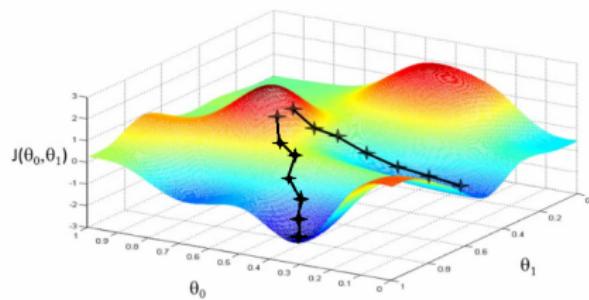
Чем алгоритмы машинного обучения отличаются друг от друга?

Какими свойствами должна обладать функция потерь?

Почему машинное обучение > оптимизация функции потерь?

# ML 101: Градиентный спуск

- $\theta$  — вектор параметров модели
- $J(\theta) : \mathbb{R}^n \rightarrow \mathbb{R}$  — функция ошибки
- $\nabla_{\theta} J(\theta) = \left( \frac{\partial}{\partial \theta_i} J(\theta) \right)_{i=1}^n$  — градиент функции ошибки
- Алгоритм: повторять, пока метод не сойдётся:
  - $\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta)$
- Параметры метода:
  - $\theta_0$  — начальная точка
  - $\eta$  — темп обучения (learning rate)
  - условие остановки



# ML 101: Метод Ньютона

$\mathbf{H}(f)(\mathbf{x})_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(\mathbf{x})$  — гессиан функции  $f(\mathbf{x})$

Ряд Тейлора в окрестности  $\mathbf{x}^{(0)}$ :

$$f(\mathbf{x}) = f(\mathbf{x}^{(0)}) + (\mathbf{x} - \mathbf{x}^{(0)})^\top \nabla_{\mathbf{x}} f(\mathbf{x}^{(0)}) + \frac{1}{2} (\mathbf{x} - \mathbf{x}^{(0)})^\top \mathbf{H}(f)(\mathbf{x}^{(0)}) (\mathbf{x} - \mathbf{x}^{(0)})$$

Повторять, пока метод не сойдётся:

- $\mathbf{x}^* = \mathbf{x}^{(0)} - \mathbf{H}(f)(\mathbf{x}^{(0)})^{-1} \nabla_{\mathbf{x}} f(\mathbf{x}^{(0)})$

# ML 101: Метод Ньютона

$\mathbf{H}(f)(\mathbf{x})_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(\mathbf{x})$  — гессиан функции  $f(\mathbf{x})$

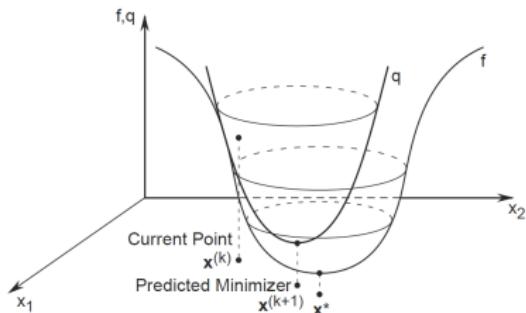
Ряд Тейлора в окрестности  $\mathbf{x}^{(0)}$ :

$$f(\mathbf{x}) = f(\mathbf{x}^{(0)}) + (\mathbf{x} - \mathbf{x}^{(0)})^\top \nabla_{\mathbf{x}} f(\mathbf{x}^{(0)}) + \frac{1}{2} (\mathbf{x} - \mathbf{x}^{(0)})^\top \mathbf{H}(f)(\mathbf{x}^{(0)}) (\mathbf{x} - \mathbf{x}^{(0)})$$

Повторять, пока метод не сойдётся:

- $\mathbf{x}^* = \mathbf{x}^{(0)} - \mathbf{H}(f)(\mathbf{x}^{(0)})^{-1} \nabla_{\mathbf{x}} f(\mathbf{x}^{(0)})$

Идеино: делаем наилучший шаг, учитывающий кривизну графика функции, приближенного квадратичной поверхностью.



# Проблемы классических методов оптимизации

Градиентный спуск:

- «Застрение» в локальном минимуме
- Гарантии сходимости только для выпуклых гладких функций
- «Зигзаги» на «оврагах»

Градиентный спуск:

- «Застрение» в локальном минимуме
- Гарантии сходимости только для выпуклых гладких функций
- «Зигзаги» на «оврагах»

Метод Ньютона:

- Использование вторых производных и ограничения на них
- «Застрение» в седловых точках

Градиентный спуск:

- «Застрение» в локальном минимуме
- Гарантии сходимости только для выпуклых гладких функций
- «Зигзаги» на «оврагах»

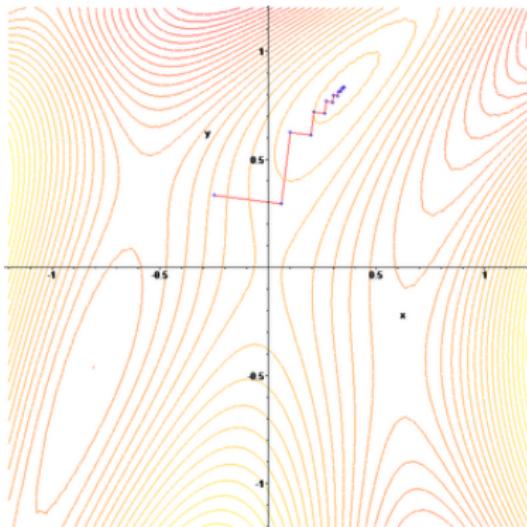
Метод Ньютона:

- Использование вторых производных и ограничения на них
- «Застрение» в седловых точках

Общая проблема:

- Много вычислений целевой функции
- Если функция вычисляется медленно, то и оптимизация медленная

# «Зигзаги» на «оврагах»



Градиентный спуск скакет по краям оврага

# ML 101: Стохастический градиентный спуск (SGD)

Смысл градиентного спуска — на каждой итерации угадать направление следующего шага.

- (в предположении, что закон вычисления длины шага фиксирован)

Смысл градиентного спуска — на каждой итерации угадать направление следующего шага.

- (в предположении, что закон вычисления длины шага фиксирован)

Идея: можно оценивать направление, исходя из поведения функции ошибки на подвыборке мощности  $k \ll n$ .

- $k = 1$  — SGD
- $k > 1$  — mini-batch gradient descent

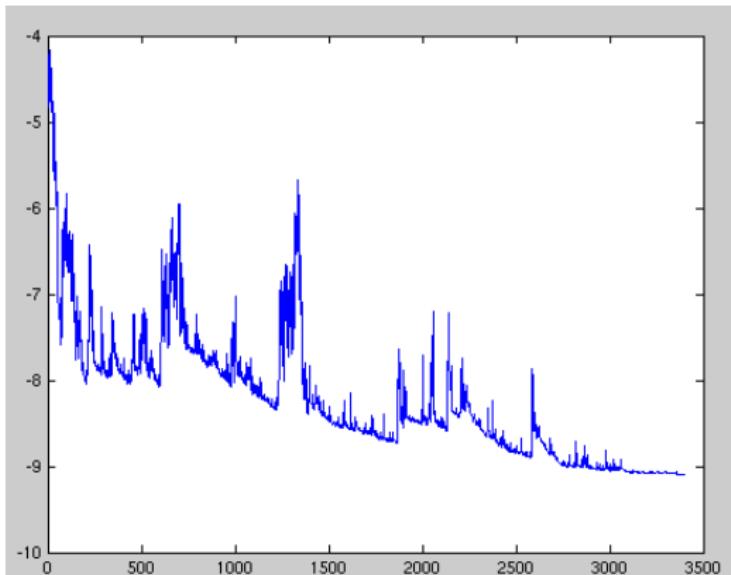
Смысл градиентного спуска — на каждой итерации угадать направление следующего шага.

- (в предположении, что закон вычисления длины шага фиксирован)

**Идея:** можно оценивать направление, исходя из поведения функции ошибки на подвыборке мощности  $k \ll n$ .

- $k = 1$  — SGD
- $k > 1$  — mini-batch gradient descent

**Теорема:** если грамотно уменьшать темп обучения, то метод когда-нибудь сойдётся туда же, куда сошёлся бы обычный градиентный спуск.



SGD: значение функции ошибок vs количество итераций

# Mini-batch gradient descent: выбор размера батча и темпа обучения

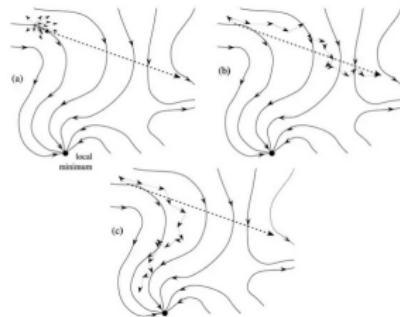
# Mini-batch gradient descent: выбор размера батча и темпа обучения

Маленькие батчи:

- «Шумные» оценки градиента
- ... но это не всегда плохо!
- Существенная экономия времени на ранних итерациях

Большие батчи:

- Более эффективное распараллеливание
- Меньший темп обучения



# ML 101: Обратное распространение ошибки

## Терминология:

- *forward pass*: движение по нейросети от входных данных к вычислению функции ошибки
- *back pass*: движение в обратном направлении

## Терминология:

- *forward pass*: движение по нейросети от входных данных к вычислению функции ошибки
- *back pass*: движение в обратном направлении
- *backpropagation*: способ вычисления градиентов для всех параметров во время back pass
- *chain rule*: правило вычисления производной сложной функции

## Терминология:

- *forward pass*: движение по нейросети от входных данных к вычислению функции ошибки
- *back pass*: движение в обратном направлении
- *backpropagation*: способ вычисления градиентов для всех параметров во время back pass
- *chain rule*: правило вычисления производной сложной функции

Производная сложной функции  $z = f(g(x)) = f(y)$ :

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$$

Многомерный случай:

$$\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n, g : \mathbb{R}^m \rightarrow \mathbb{R}^n, f : \mathbb{R}^n \rightarrow \mathbb{R}$$

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

# Backprop: symbol-to-number



Figure 6.9: A computational graph that results in repeated subexpressions when computing the gradient. Let  $w \in \mathbb{R}$  be the input to the graph. We use the same function  $f : \mathbb{R} \rightarrow \mathbb{R}$  as the operation that we apply at every step of a chain:  $x = f(w)$ ,  $y = f(x)$ ,  $z = f(y)$ . To compute  $\frac{\partial z}{\partial w}$ , we apply Eq. 6.44 and obtain:

$$\frac{\partial z}{\partial w} \quad (6.50)$$

$$= \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \frac{\partial x}{\partial w} \quad (6.51)$$

$$= f'(y) f'(x) f'(w) \quad (6.52)$$

$$= f'(f(f(w))) f'(f(f(w))) f'(w) \quad (6.53)$$

# Backprop: symbol-to-symbol

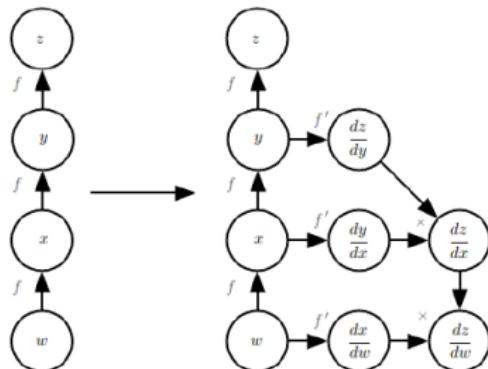


Figure 6.10: An example of the symbol-to-symbol approach to computing derivatives. In this approach, the back-propagation algorithm does not need to ever access any actual specific numeric values. Instead, it adds nodes to a computational graph describing how to compute these derivatives. A generic graph evaluation engine can later compute the derivatives for any specific numeric values. (Left) In this example, we begin with a graph representing  $z = f(f(f(w)))$ . (Right) We run the back-propagation algorithm, instructing it to construct the graph for the expression corresponding to  $\frac{dz}{dw}$ . In this example, we do not explain how the back-propagation algorithm works. The purpose is only to illustrate what the desired result is: a computational graph with a symbolic description of the derivative.

# Трудности оптимизации в обучении нейронных сетей

Специфика:

- Плохая обусловленность гессиана  $H$

Специфика:

- Плохая обусловленность гессиана  $H$
- Множество локальных минимумов функции ошибки

Специфика:

- Плохая обусловленность гессиана  $H$
- Множество локальных минимумов функции ошибки
- Плато, седловые точки, утёсы

## Специфика:

- Плохая обусловленность гессиана  $H$
- Множество локальных минимумов функции ошибки
- Плато, седловые точки, утёсы
- Несоответствие между локальной и глобальной структурой поверхности

Специфика:

- Плохая обусловленность гессиана  $H$
- Множество локальных минимумов функции ошибки
- Плато, седловые точки, утёсы
- Несоответствие между локальной и глобальной структурой поверхности
- ... и многое, многое другое!

# Плохая обусловленность гессиана

## Плохая обусловленность матрицы $H$

Число обусловленности —  $\max_{i,j} \left| \frac{\lambda_i}{\lambda_j} \right|$ , где  $\{\lambda_i\}$  — множество собственных значений матрицы  $H$ .

Плохая обусловленность = большое значение числа обусловленности.

## Плохая обусловленность матрицы $H$

Число обусловленности —  $\max_{i,j} \left| \frac{\lambda_i}{\lambda_j} \right|$ , где  $\{\lambda_i\}$  — множество собственных значений матрицы  $H$ .

Плохая обусловленность = большое значение числа обусловленности.

Почему нам важен гессиан?

## Плохая обусловленность матрицы $H$

Число обусловленности —  $\max_{i,j} |\frac{\lambda_i}{\lambda_j}|$ , где  $\{\lambda_i\}$  — множество собственных значений матрицы  $H$ .

Плохая обусловленность = большое значение числа обусловленности.

Почему нам важен гессиан?

- Ряд Тейлора

$$J(\boldsymbol{\theta} - \eta \cdot \nabla_{\boldsymbol{\theta}}) \approx J(\boldsymbol{\theta}) - \eta \cdot \nabla_{\boldsymbol{\theta}}^\top \nabla_{\boldsymbol{\theta}} + \frac{1}{2} \eta^2 \nabla_{\boldsymbol{\theta}}^\top H \nabla_{\boldsymbol{\theta}}$$

## Плохая обусловленность матрицы $H$

Число обусловленности —  $\max_{i,j} \left| \frac{\lambda_i}{\lambda_j} \right|$ , где  $\{\lambda_i\}$  — множество собственных значений матрицы  $H$ .

Плохая обусловленность = большое значение числа обусловленности.

Почему нам важен гессиан?

- Ряд Тейлора

$$J(\boldsymbol{\theta} - \eta \cdot \nabla_{\boldsymbol{\theta}}) \approx J(\boldsymbol{\theta}) - \eta \cdot \nabla_{\boldsymbol{\theta}}^\top \nabla_{\boldsymbol{\theta}} + \frac{1}{2} \eta^2 \nabla_{\boldsymbol{\theta}}^\top H \nabla_{\boldsymbol{\theta}}$$

Как отследить подобную ситуацию?

## Плохая обусловленность матрицы $H$

Число обусловленности —  $\max_{i,j} |\frac{\lambda_i}{\lambda_j}|$ , где  $\{\lambda_i\}$  — множество собственных значений матрицы  $H$ .

Плохая обусловленность = большое значение числа обусловленности.

Почему нам важен гессиан?

- Ряд Тейлора

$$J(\theta - \eta \cdot \nabla_\theta) \approx J(\theta) - \eta \cdot \nabla_\theta^\top \nabla_\theta + \frac{1}{2} \eta^2 \nabla_\theta^\top H \nabla_\theta$$

Как отследить подобную ситуацию?

- Мониторить значение  $\nabla_\theta^\top \nabla_\theta$
- Мониторить значение  $\nabla_\theta^\top H \nabla_\theta$

# Плохая обусловленность гессиана

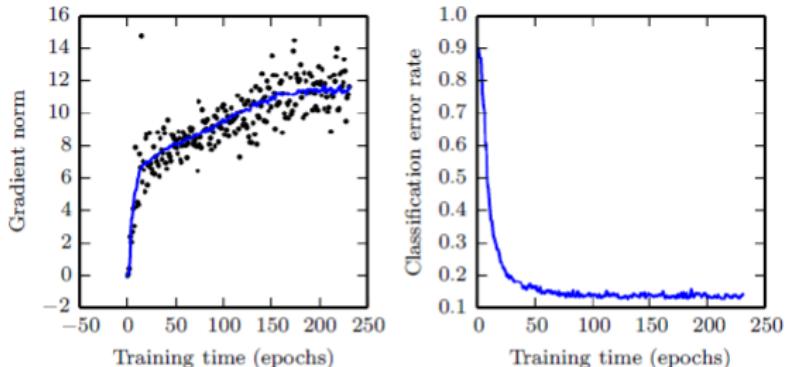


Figure 8.1: Gradient descent often does not arrive at a critical point of any kind. In this example, the gradient norm increases throughout training of a convolutional network used for object detection. (*Left*) A scatterplot showing how the norms of individual gradient evaluations are distributed over time. To improve legibility, only one gradient norm is plotted per epoch. The running average of all gradient norms is plotted as a solid curve. The gradient norm clearly increases over time, rather than decreasing as we would expect if the training process converged to a critical point. (*Right*) Despite the increasing gradient, the training process is reasonably successful. The validation set classification error decreases to a low level.

# Локальные минимумы, плато, седловые точки

Откуда берутся локальные минимумы?

Откуда берутся локальные минимумы?

- невыпуклая функция ошибки
- симметрия пространства параметров
- часто: возможность масштабирования параметров

Откуда берутся локальные минимумы?

- невыпуклая функция ошибки
- симметрия пространства параметров
- часто: возможность масштабирования параметров

Почему много седловых точек?

Откуда берутся локальные минимумы?

- невыпуклая функция ошибки
- симметрия пространства параметров
- часто: возможность масштабирования параметров

Почему много седловых точек?

- В седловых точках **не все** собственные значения гессиана одного знака.
- В точках экстремума **все** собственные значения гессиана одного знака.

Откуда берутся локальные минимумы?

- невыпуклая функция ошибки
- симметрия пространства параметров
- часто: возможность масштабирования параметров

Почему много седловых точек?

- В седловых точках **не все** собственные значения гессиана одного знака.
- В точках экстремума **все** собственные значения гессиана одного знака.

Как появляются плато?

Откуда берутся локальные минимумы?

- невыпуклая функция ошибки
- симметрия пространства параметров
- часто: возможность масштабирования параметров

Почему много седловых точек?

- В седловых точках **не все** собственные значения гессиана одного знака.
- В точках экстремума **все** собственные значения гессиана одного знака.

Как появляются плато?

- 0–1 loss
- saturated units
- dead units

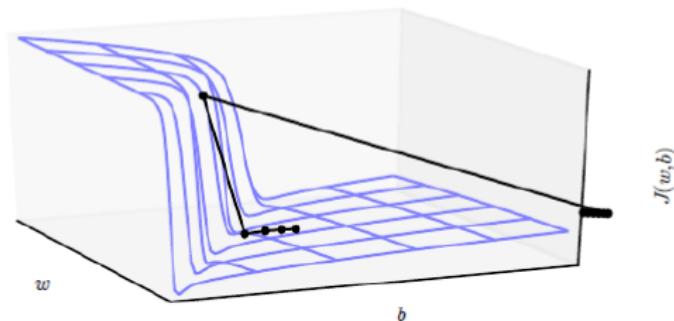


Figure 8.3: The objective function for highly nonlinear deep neural networks or for recurrent neural networks often contains sharp nonlinearities in parameter space resulting from the multiplication of several parameters. These nonlinearities give rise to very high derivatives in some places. When the parameters get close to such a cliff region, a gradient descent update can catapult the parameters very far, possibly losing most of the optimization work that had been done. Figure adapted with permission from [Pascanu et al. \(2013a\)](#).

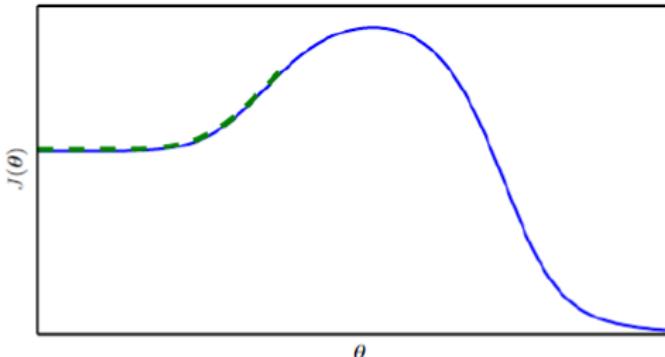


Figure 8.4: Optimization based on local downhill moves can fail if the local surface does not point toward the global solution. Here we provide an example of how this can occur, even if there are no saddle points and no local minima. This example cost function contains only asymptotes toward low values, not minima. The main cause of difficulty in this case is being initialized on the wrong side of the “mountain” and not being able to traverse it. In higher dimensional space, learning algorithms can often circumnavigate such mountains but the trajectory associated with doing so may be long and result in excessive training time, as illustrated in Fig. 8.2.

# Алгоритмы: Momentum

Идея: оптимизация — движение точки, обладающей массой.

Идея: оптимизация — движение точки, обладающей массой.

- $v_t = \gamma v_{t-1} + \eta \cdot \nabla_{\theta} J(\theta)$
- $\theta = \theta - v_t$
- $\gamma \in [0, 1)$

Идея: оптимизация — движение точки, обладающей массой.

- $v_t = \gamma v_{t-1} + \eta \cdot \nabla_{\theta} J(\theta)$
- $\theta = \theta - v_t$
- $\gamma \in [0, 1]$

«Скорость» накапливает информацию о предыдущих значениях градиента. Это помогает избежать осцилляций (например, на стенках оврага).

На практике  $\gamma \in [0.5, 0.99]$  и может меняться со временем.

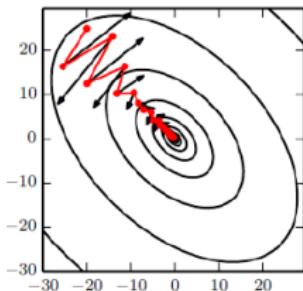


Figure 8.5: Momentum aims primarily to solve two problems: poor conditioning of the Hessian matrix and variance in the stochastic gradient. Here, we illustrate how momentum overcomes the first of these two problems. The contour lines depict a quadratic loss function with a poorly conditioned Hessian matrix. The red path cutting across the contours indicates the path followed by the momentum learning rule as it minimizes this function. At each step along the way, we draw an arrow indicating the step that gradient descent would take at that point. We can see that a poorly conditioned quadratic objective looks like a long, narrow valley or canyon with steep sides. Momentum correctly traverses the canyon lengthwise, while gradient steps waste time moving back and forth across the narrow axis of the canyon. Compare also Fig. 4.6, which shows the behavior of gradient descent without momentum.

# Алгоритмы: Nesterov Momentum

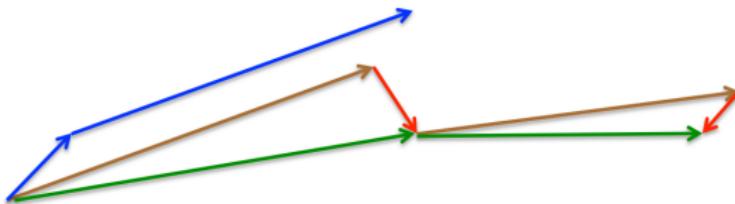
**Идея:** «заглядываем вперёд», в точку, куда попадём из-за накопленного импульса. Вычисляем градиент в этой точке.

**Идея:** «заглядываем вперёд», в точку, куда попадём из-за накопленного импульса. Вычисляем градиент в этой точке.

- $v_t = \gamma v_{t-1} + \eta \cdot \nabla_{\theta} J(\theta - \gamma v_{t-1})$
- $\theta = \theta - v_t$
- Momentum:  $v_t = \gamma v_{t-1} + \eta \cdot \nabla_{\theta} J(\theta)$

**Идея:** «заглядываем вперёд», в точку, куда попадём из-за накопленного импульса. Вычисляем градиент в этой точке.

- $\mathbf{v}_t = \gamma \mathbf{v}_{t-1} + \eta \cdot \nabla_{\theta} J(\theta - \gamma \mathbf{v}_{t-1})$
- $\theta = \theta - \mathbf{v}_t$
- Momentum:  $\mathbf{v}_t = \gamma \mathbf{v}_{t-1} + \eta \cdot \nabla_{\theta} J(\theta)$



# Nesterov Momentum

# Nesterov Momentum

# Nesterov Momentum

# Алгоритмы: AdaGrad

**Идея:** зададим собственный темп обучения для каждого параметра  $\theta_i$ . Если параметр сильно изменяется во время оптимизации, будем делать шаги меньшей длины, и наоборот.

**Идея:** зададим собственный темп обучения для каждого параметра  $\theta_i$ . Если параметр сильно изменяется во время оптимизации, будем делать шаги меньшей длины, и наоборот.

- $g_{t,i} = \nabla_{\theta} J(\theta_i)$
- SGD:  $\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$
- AdaGrad:  $\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$
- $G_t$  — диагональная матрица. На  $i$ -ом месте — сумма квадратов всех градиентов по  $\theta_i$  до  $t$ -ой итерации алгоритма.

**Идея:** зададим собственный темп обучения для каждого параметра  $\theta_i$ . Если параметр сильно изменяется во время оптимизации, будем делать шаги меньшей длины, и наоборот.

- $g_{t,i} = \nabla_{\theta} J(\theta_i)$
- SGD:  $\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$
- AdaGrad:  $\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$
- $G_t$  — диагональная матрица. На  $i$ -ом месте — сумма квадратов всех градиентов по  $\theta_i$  до  $t$ -ой итерации алгоритма.

**Проблема:** AdaGrad «помнит» всю историю предыдущих итераций, поэтому величина шага всё время уменьшается.

# Алгоритмы: RMSProp и AdaDelta

**Идея:** постепенно «забываем» старые данные о величинах градиентов.

Идея: постепенно «забываем» старые данные о величинах градиентов.

- $E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma)g_t^2$
- RMSProp:  $\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{E[g^2]_{t,ii} + \epsilon}} \cdot g_{t,i}$

**Идея:** постепенно «забываем» старые данные о величинах градиентов.

- $E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma)g_t^2$
- RMSProp:  $\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{E[g^2]_{t,ii} + \varepsilon}} \cdot g_{t,i}$
- $\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \varepsilon}} \cdot g_t = -\frac{\eta}{RMS[g]_t} \cdot g_t$
- $E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1 - \gamma)\Delta\theta_t^2$
- AdaDelta:  $\theta_{t+1,i} = \theta_{t,i} - \frac{RMS[\Delta\theta]_{t-1,i}}{RMS[g]_{t,i}} \cdot g_{t,i}$

# Алгоритмы: Adam, AdaMax, Nadam

**Adam:** RMSProp + Momentum

## Adam: RMSProp + Momentum

- $m_t = \beta_1 m_{t-1} + (1 - \beta_1)g_t$
- $v_t = \beta_2 v_{t-1} + (1 - \beta_2)g_t^2$
- $\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \hat{v}_t = \frac{v_t}{1-\beta_2^t}$
- $\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{\hat{v}_{t,i} + \epsilon}} \cdot \hat{m}_{t,i}$

## Adam: RMSProp + Momentum

- $m_t = \beta_1 m_{t-1} + (1 - \beta_1)g_t$
- $v_t = \beta_2 v_{t-1} + (1 - \beta_2)g_t^2$
- $\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \hat{v}_t = \frac{v_t}{1-\beta_2^t}$
- $\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{\hat{v}_{t,i} + \epsilon}} \cdot \hat{m}_{t,i}$

AdaMax: вместо  $\ell_2$ -нормы используем  $\ell_\infty$ -норму.

## Adam: RMSProp + Momentum

- $m_t = \beta_1 m_{t-1} + (1 - \beta_1)g_t$
- $v_t = \beta_2 v_{t-1} + (1 - \beta_2)g_t^2$
- $\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \hat{v}_t = \frac{v_t}{1-\beta_2^t}$
- $\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{\hat{v}_{t,i} + \epsilon}} \cdot \hat{m}_{t,i}$

AdaMax: вместо  $\ell_2$ -нормы используем  $\ell_\infty$ -норму.

## Nadam: RMSProp + Nesterov Momentum

# Сравнение алгоритмов

# Алгоритмы: BFGS, LBFGS

**BFGS:** не будем вычислять  $H^{-1}$ , а будем строить для этой матрицы приближение на каждой итерации работы алгоритма.

**BFGS:** не будем вычислять  $H^{-1}$ , а будем строить для этой матрицы приближение на каждой итерации работы алгоритма.

- Шаг в методе Ньютона:  $\theta = \theta - H^{-1} \nabla_{\theta} J(\theta)$
- Нужно хранить матрицу  $O(n^2)$

**BFGS:** не будем вычислять  $H^{-1}$ , а будем строить для этой матрицы приближение на каждой итерации работы алгоритма.

- Шаг в методе Ньютона:  $\theta = \theta - H^{-1} \nabla_{\theta} J(\theta)$
- Нужно хранить матрицу  $O(n^2)$

**Limited Memory BFGS:** не храним приближение для  $H^{-1}$ , а вычисляем приближение для приближения на лету.



# Выводы

Какой алгоритм лучше?

Какой алгоритм лучше?

Ответ: что такое «лучшее»?

- Проще всего в реализации: **SGD**

Какой алгоритм лучше?

Ответ: что такое «лучшее»?

- Проще всего в реализации: **SGD**
- Экономная траектория: методы с адаптивным темпом обучения

Какой алгоритм лучше?

Ответ: что такое «лучше»?

- Проще всего в реализации: **SGD**
- Экономная траектория: методы с адаптивным темпом обучения
- Не нужно настраивать гиперпараметры: **AdaDelta**

Какой алгоритм лучше?

Ответ: что такое «лучше»?

- Проще всего в реализации: **SGD**
- Экономная траектория: методы с адаптивным темпом обучения
- Не нужно настраивать гиперпараметры: **AdaDelta**
- Оптимальный выбор для работы «из коробки»: **Adam**

Какой алгоритм лучше?

Ответ: что такое «лучше»?

- Проще всего в реализации: **SGD**
- Экономная траектория: методы с адаптивным темпом обучения
- Не нужно настраивать гиперпараметры: **AdaDelta**
- Оптимальный выбор для работы «из коробки»: **Adam**
- Потенциально лучшие минимумы: **L-BFGS**

# Ключевые слова для дополнительного чтения

- Exploding and vanishing gradients
- Early stopping
- Multistart
- Curriculum learning
- Batchnorm
- Non-gradient optimization (differential evolution etc.)

Спасибо за внимание!  
[alexey.romanov@phystech.edu](mailto:alexey.romanov@phystech.edu)