Методы стохастической оптимизации

Беляков Денис, ПМИ ФКН 152
Артемьев Максим, ПМИ ФКН 152

Содержание:

- Варианты градиентного спуска
- SGD + momentum, NAG
- Adagrad, Adadelta, RMSprop
- SAG

Gradient descent variants

- Vanilla (aka Batch) Gradient Descent
- Stochastic Gradient Descent
- Mini-batch Gradient Descent

Batch Gradient Descent

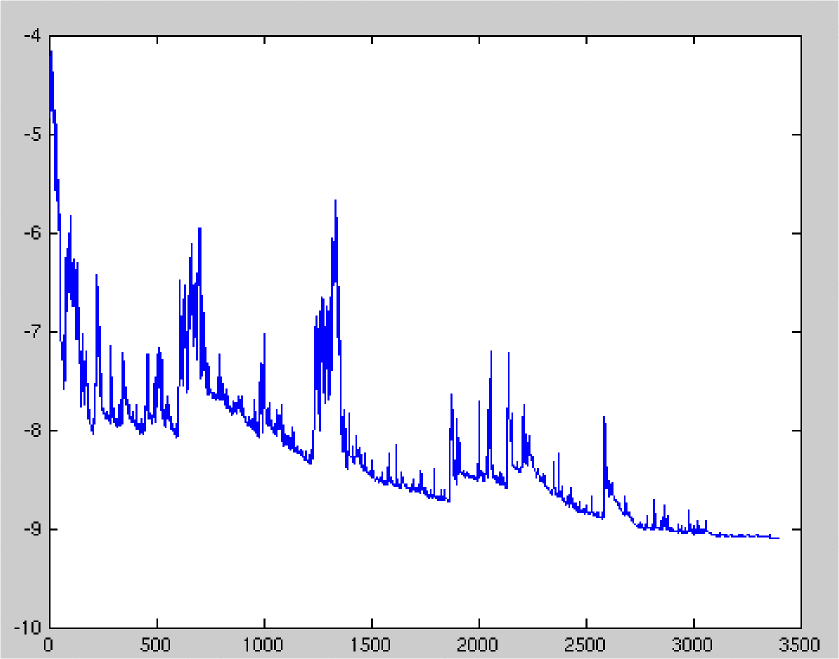
```
for i in range(nb_epochs):
  params_grad = evaluate_gradient(loss_function, data, params)
  params = params - learning_rate * params_grad
```

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

Θ – параметры
η – длина шага / learning rate
J – значение функционала ошибки

Stochastic Gradient Descent

```python
for i in range(nb_epochs):
  np.random.shuffle(data)
  for example in data:
    params_grad = evaluate_gradient(loss_function, example, params)
    params = params - learning_rate * params_grad
```



Wikipedia

$$\theta = \theta - \eta \cdot \nabla_\theta \mathbf{J}(\theta; \mathbf{x}^{(i)}; \mathbf{y}^{(i)})$$

Θ – параметры
η – длина шага / learning rate
J – значение функционала ошибки
$x^{(i)}, y^{(i)}$ - случайно взятый элемент из обучающей выборки и его label

Mini-batch Gradient Descent

```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for batch in get_batches(data, batch_size=50):
    params_grad = evaluate_gradient(loss_function, batch, params)
    params = params - learning_rate * params_grad
```

Итерируемся не по случайно взятым элементам, а по мини-батчам

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

Θ – параметры
η – длина шага / learning rate
J – значение функционала ошибки
$x^{(i:i+n)}$, $y^{(i:i+n)}$ - случайно взятые элементы из обучающей выборки и их labels

Проблемы градиентного спуска

- Необходимость выбора правильного параметра размера шага
  - *too small* → *too slow*
  - *too large* → *fluctuations around minimum or even divergence*
- Застревание в седловых точках и локальных минимумах
- Размер шага является одинаковым для всех параметров

SGD with Momentum

$$v_t = \gamma v_{t-1} + (1 - \gamma)x$$

Экспоненциальное скользящее среднее

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

Использование накопленного градиента

$$\theta = \theta - v_t$$

Изменение параметров

Дефолтное значение \gamma = 0.9

Nesterov accelerated gradient

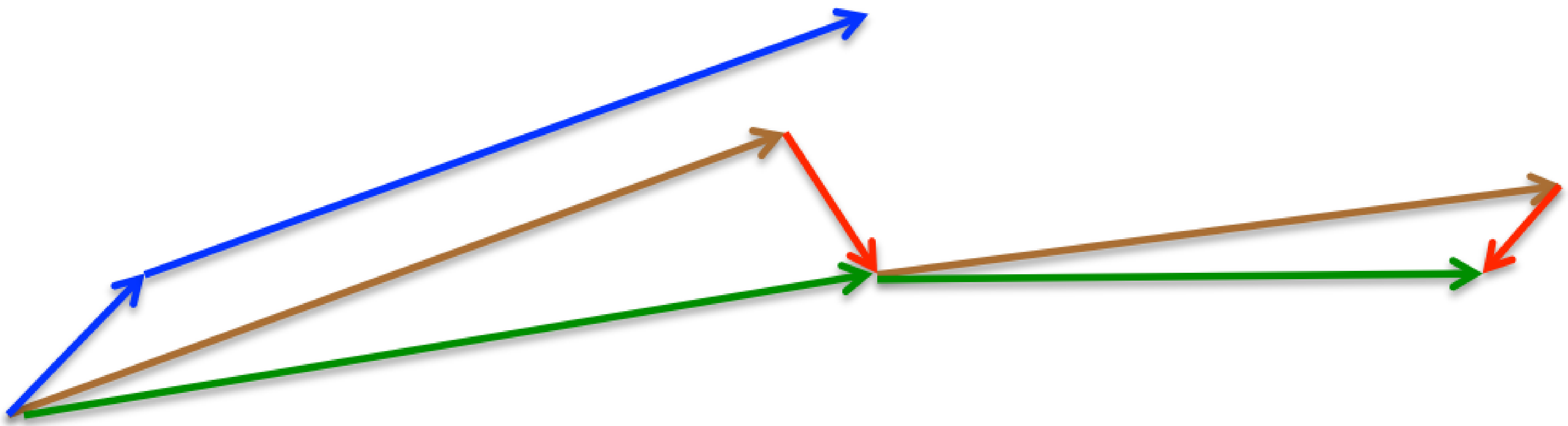
Посчитать градиент функции потерь в точке $\theta - \gamma v_{t-1}$

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

$$\theta = \theta - v_t$$

Изменение параметров

Nesterov differences



SGD + momentum - синий вектор

NAG - коричневый + красный = зеленый

Adagrad

$g_{t,i} = \nabla_\theta J(\theta_i).$ градиент функции для параметра θ_i в момент времени t

$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}.$ SGD update для каждого параметра θ_i в момент времени t

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

Adagrad изменяет размер шага в каждый момент времени для каждого параметра на основе ранее посчитанных градиентов

$G_t \in \mathbb{R}^{d \times d}$ Диагональная матрица, где каждый элемент – это сумма квадратов градиентов к моменту времени t

Adagrad

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t.$$

ϵ - небольшое значение для предотвращения деления на ноль

Главный плюс – не надо руками подбирать размер шага. Большинство имплементаций просто используют дефолтное значение 0.01

Adagrad

RMSprop

- Вместо накопления квадратов градиентов используется скользящее среднее

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2.$$

- Тогда формула обновления будет такова

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

- RMS - Root Mean Square

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}.$$

rprop: Using only the sign of the gradient

- The magnitude of the gradient can be very different for different weights and can change during learning.
  - This makes it hard to choose a single global learning rate.
- For full batch learning, we can deal with this variation by only using the sign of the gradient.
  - The weight updates are all of the same magnitude.
  - This escapes from plateaus with tiny gradients quickly.

- rprop: This combines the idea of only using the sign of the gradient with the idea of adapting the step size separately for each weight.
  - Increase the step size for a weight multiplicatively (*e.g.* times 1.2) if the signs of its last two gradients agree.
  - Otherwise decrease the step size multiplicatively (*e.g.* times 0.5).
  - Limit the step sizes to be less than 50 and more than a millionth (Mike Shuster's advice).

rmsprop: A mini-batch version of rprop

- rprop is equivalent to using the gradient but also dividing by the size of the gradient.
  - The problem with mini-batch rprop is that we divide by a different number for each mini-batch. So why not force the number we divide by to be very similar for adjacent mini-batches?
- rmsprop: Keep a moving average of the squared gradient for each weight

$$MeanSquare(w,t) = 0.9\ MeanSquare(w,\ t-1) + 0.1\left(\partial E/_{\partial w}(t)\right)^2$$

- Dividing the gradient by $\sqrt{MeanSquare(w,\ t)}$ makes the learning work much better (Tijmen Tieleman, unpublished).

Adadelta

- Расширение Adagrad'a.
- Так же как и в RMSprop используется скользящее среднее
- Добавляем в числитель стабилизирующий элемент

$$\Delta\theta = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$

- Формула обновления:

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

ADAM - adaptive moment estimation

- Накапливаем значения градиента - SGD with momentum

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

- Оцениваем среднюю нецентрированную дисперсию - то же самое что и $E[g^2]_t$

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

- Начальная калибровка для нескольких первых шагов

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

ADAM - adaptive moment estimation

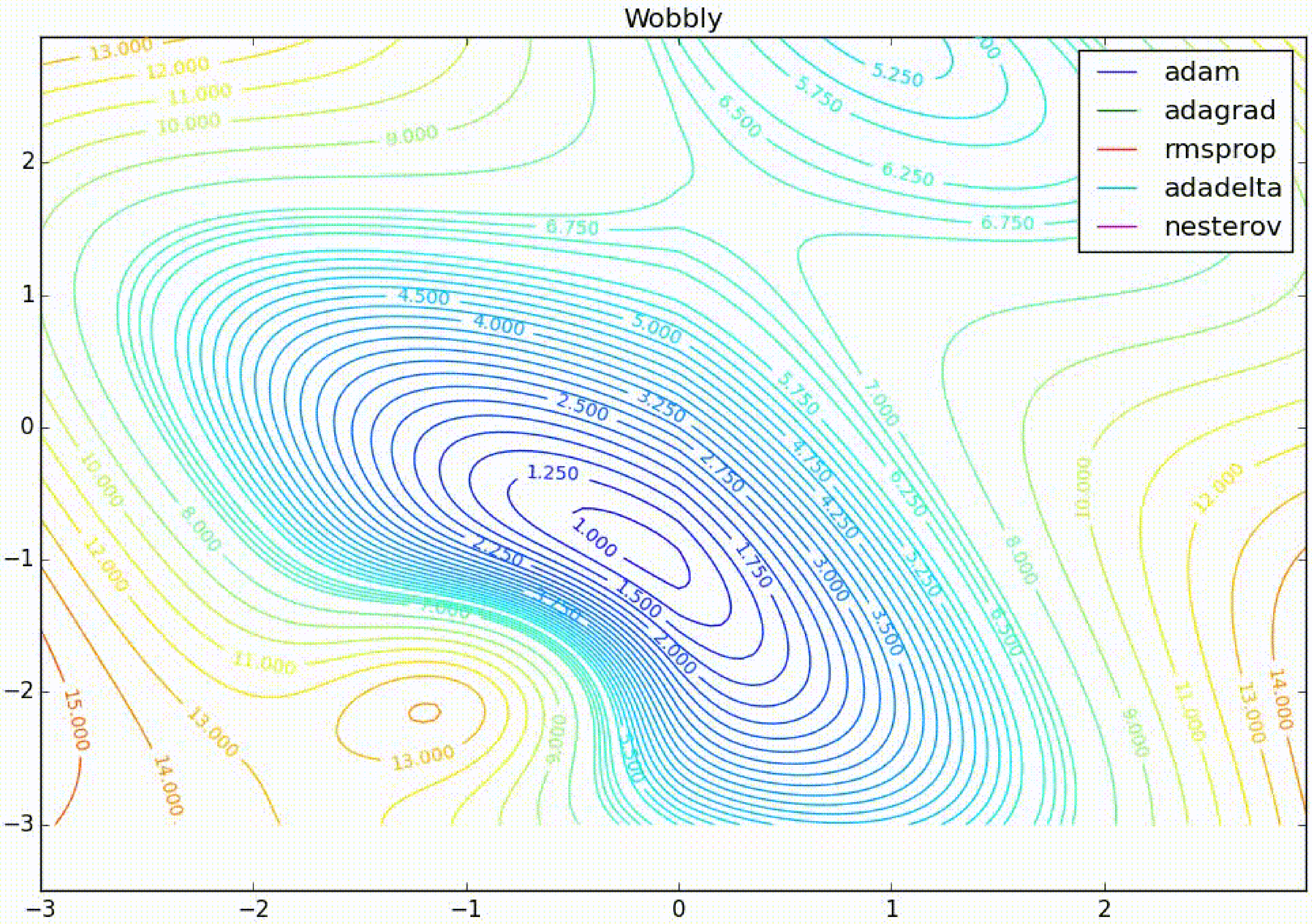
- В итоге, общее правило обновления

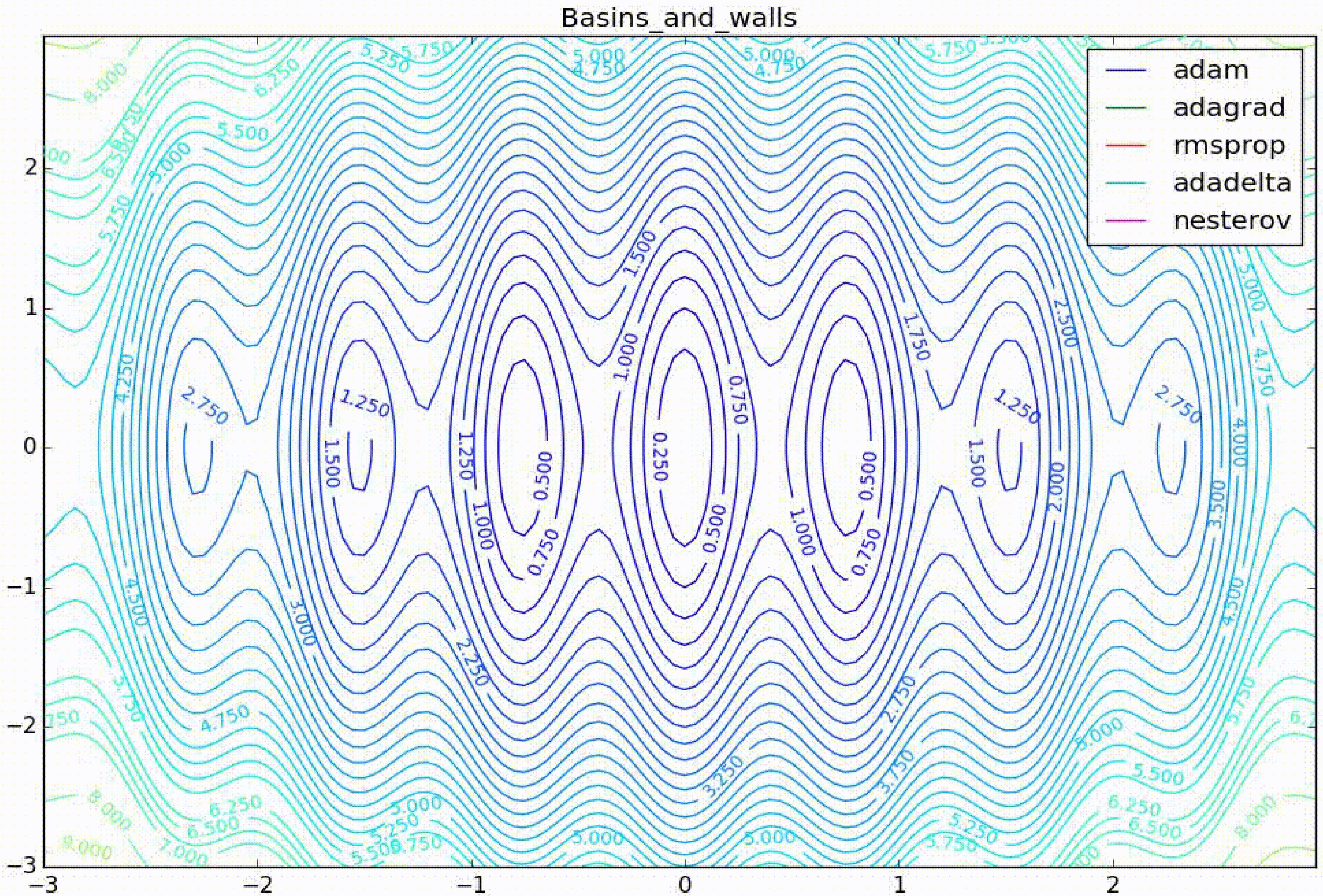
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

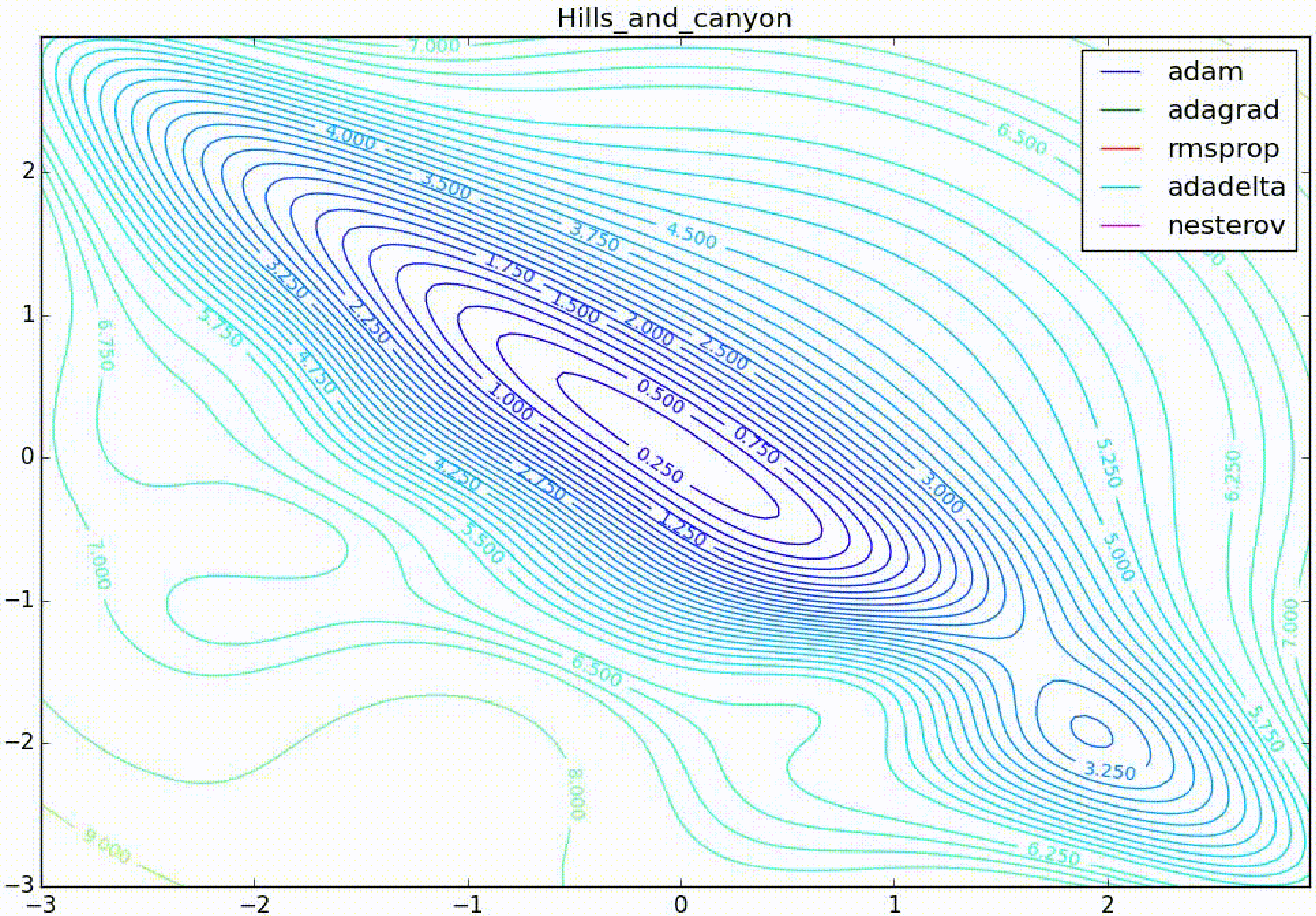
- Рекомендуются такие значение гиперпараметров

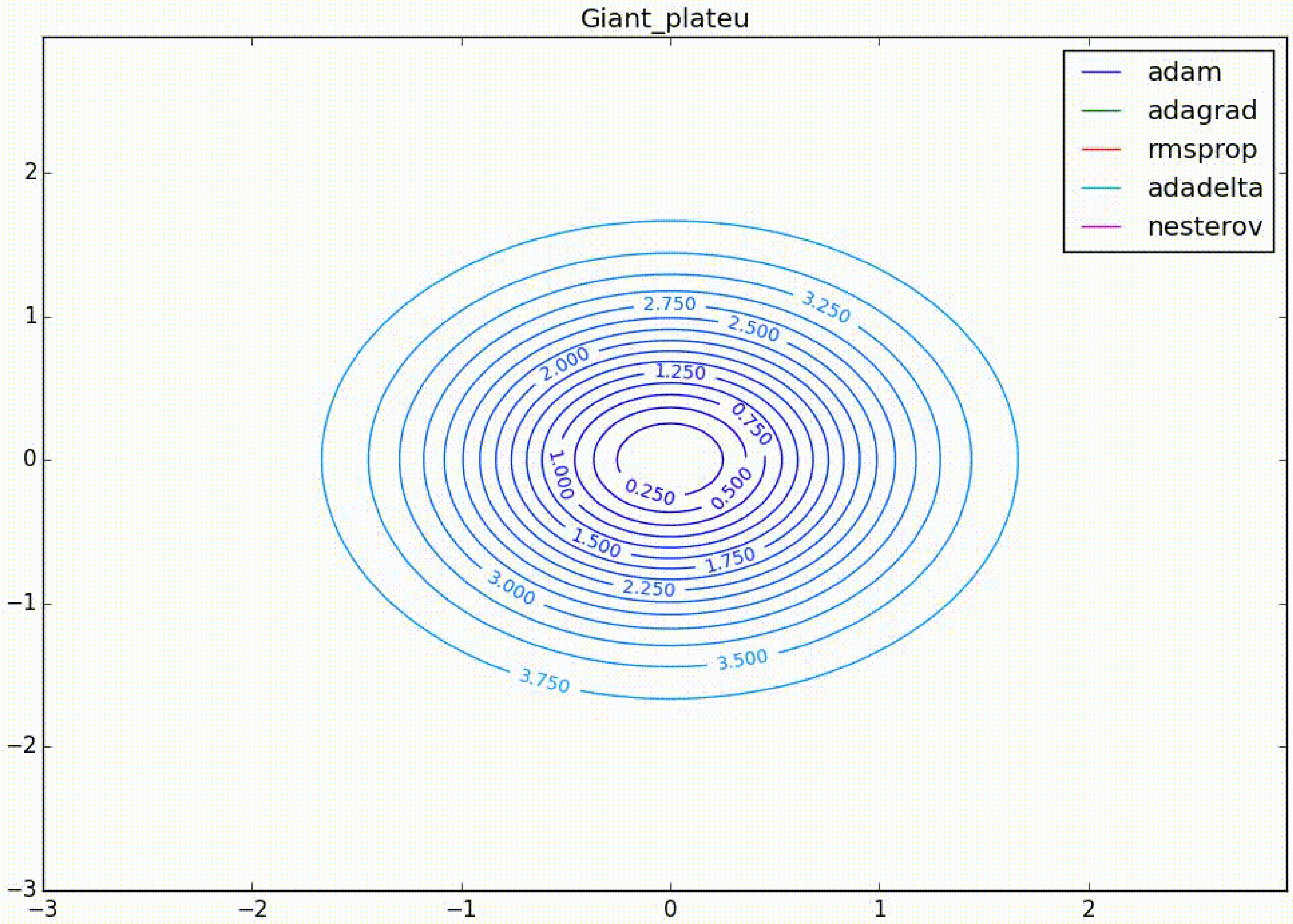
$$\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$$

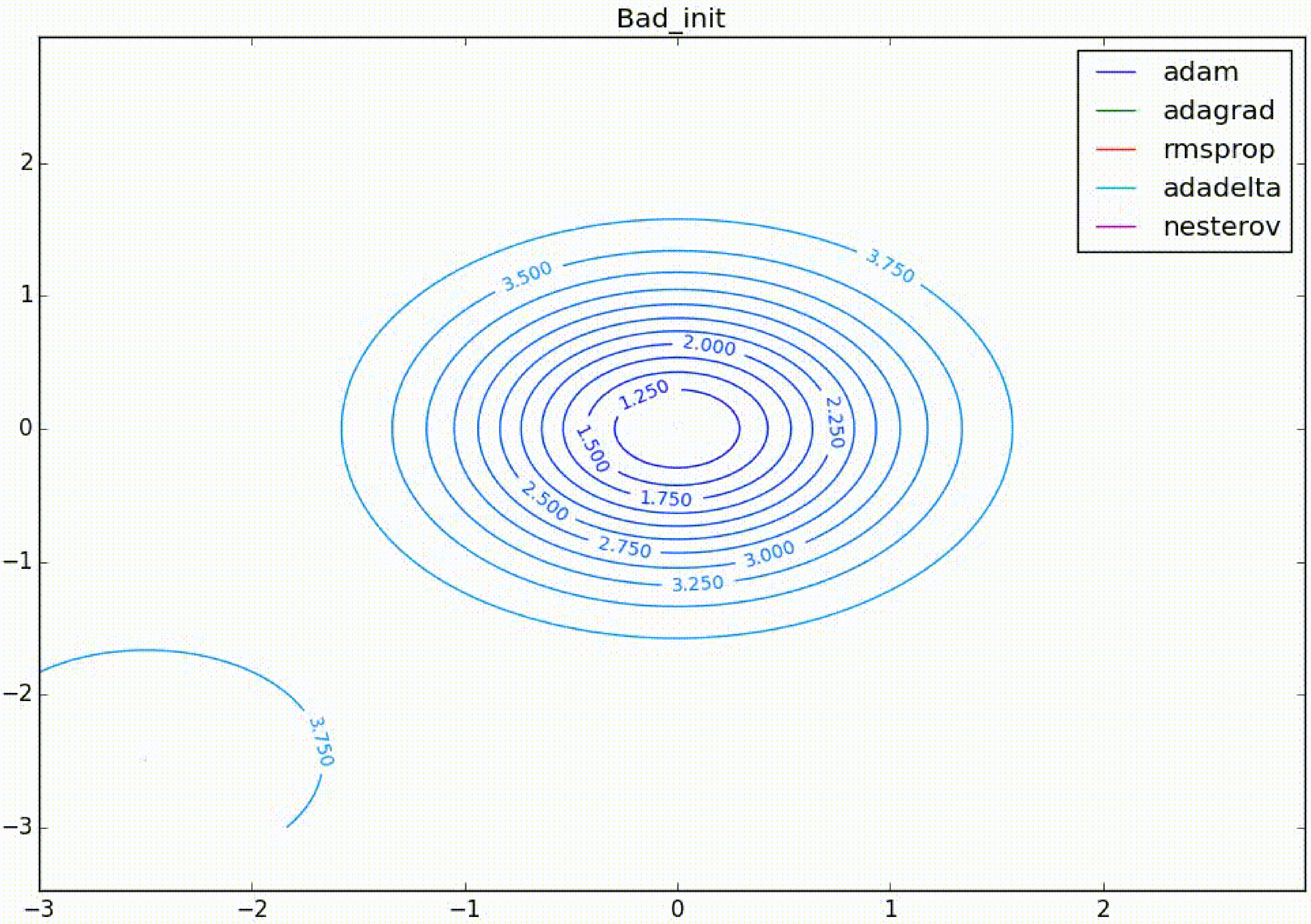
- К ADAM можно применить метод Нестерова, получится NADAM

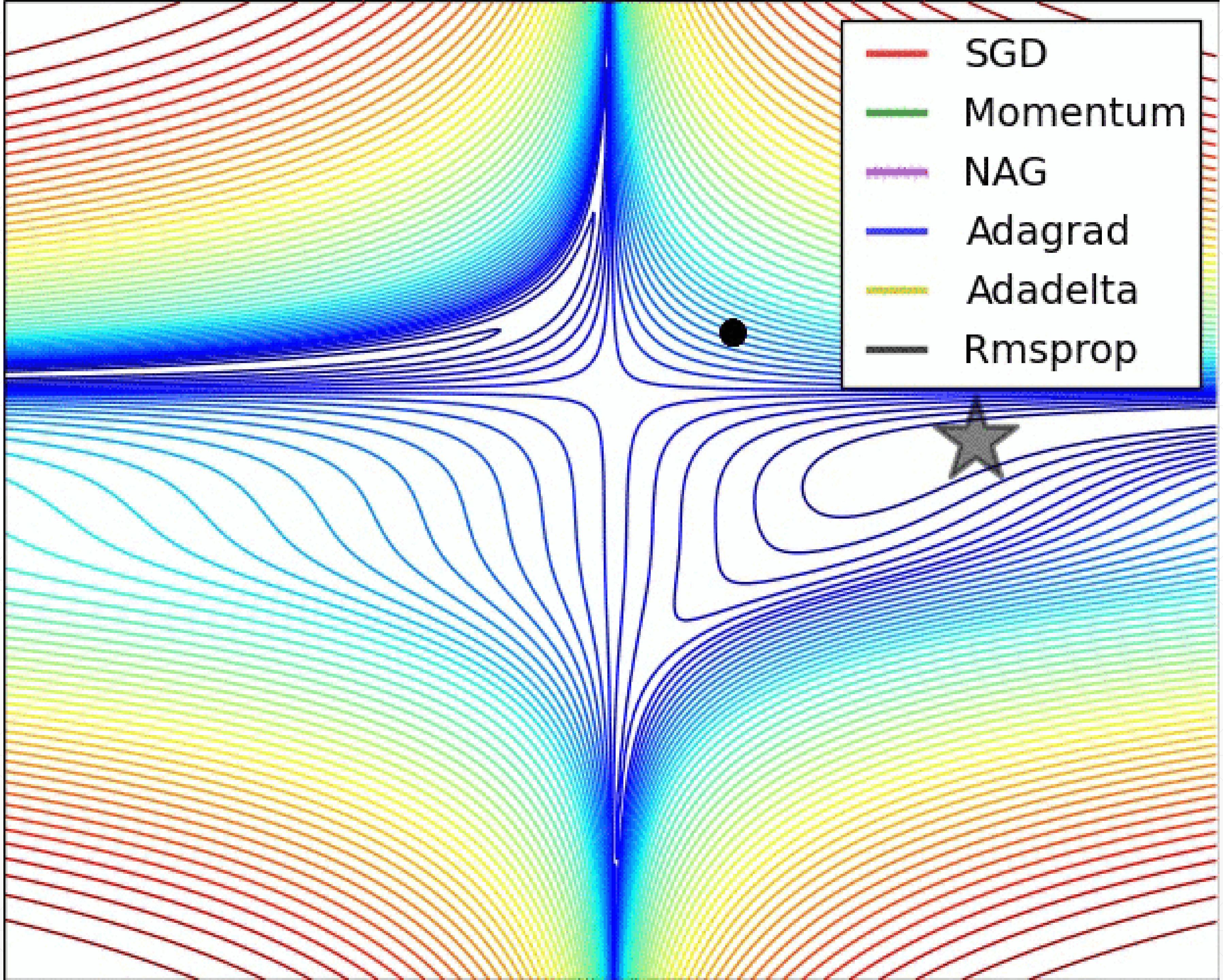

Wobbly
adam
adagrad
rmsprop
adadelta
nesterov

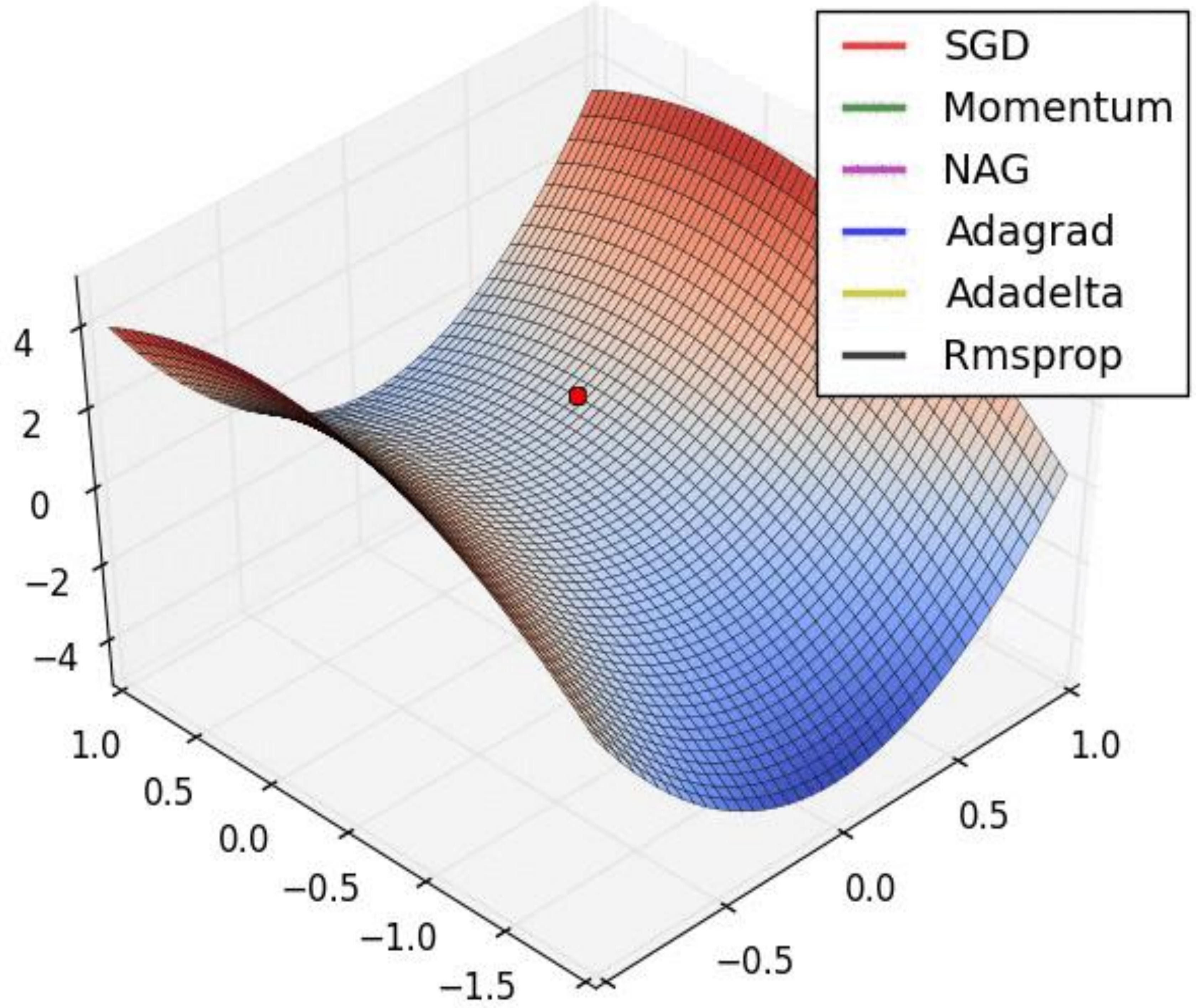

Basins_and_walls
adam
adagrad
rmsprop
adadelta
nesterov
−3
−2
−1
0
1
2


Hills_and_canyon
adam
adagrad
rmsprop
adadelta
nesterov
−3
−2
−1
0
1
2
−3
−2
−1
0
1
2


Giant_plateu
adam
adagrad
rmsprop
adadelta
nesterov
0.250
0.500
0.750
1.000
1.250
1.500
1.750
2.000
2.250
2.500
2.750
3.000
3.250
3.500
3.750
−3
−2
−1
0
1
2


Bad_init
adam
adagrad
rmsprop
adadelta
nesterov
3.750
3.500
2.000
1.250
2.250
1.500
1.750
2.500
2.750
3.000
3.250
3.750
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
1.0
0.5
0.0
−0.5

Stochastic Average Gradient

1. На вход подаются выборка, параметр темпа обучения h и параметр темпа забывания.
2. Вычисляются и держатся в памяти градиенты для каждого объекта, G_i
3. Инициализируется оценка функционала

$$\bar{Q} := \frac{1}{\ell} \sum_{i=1}^{\ell} \mathscr{L}_i(w);$$

Stochastic Average Gradient

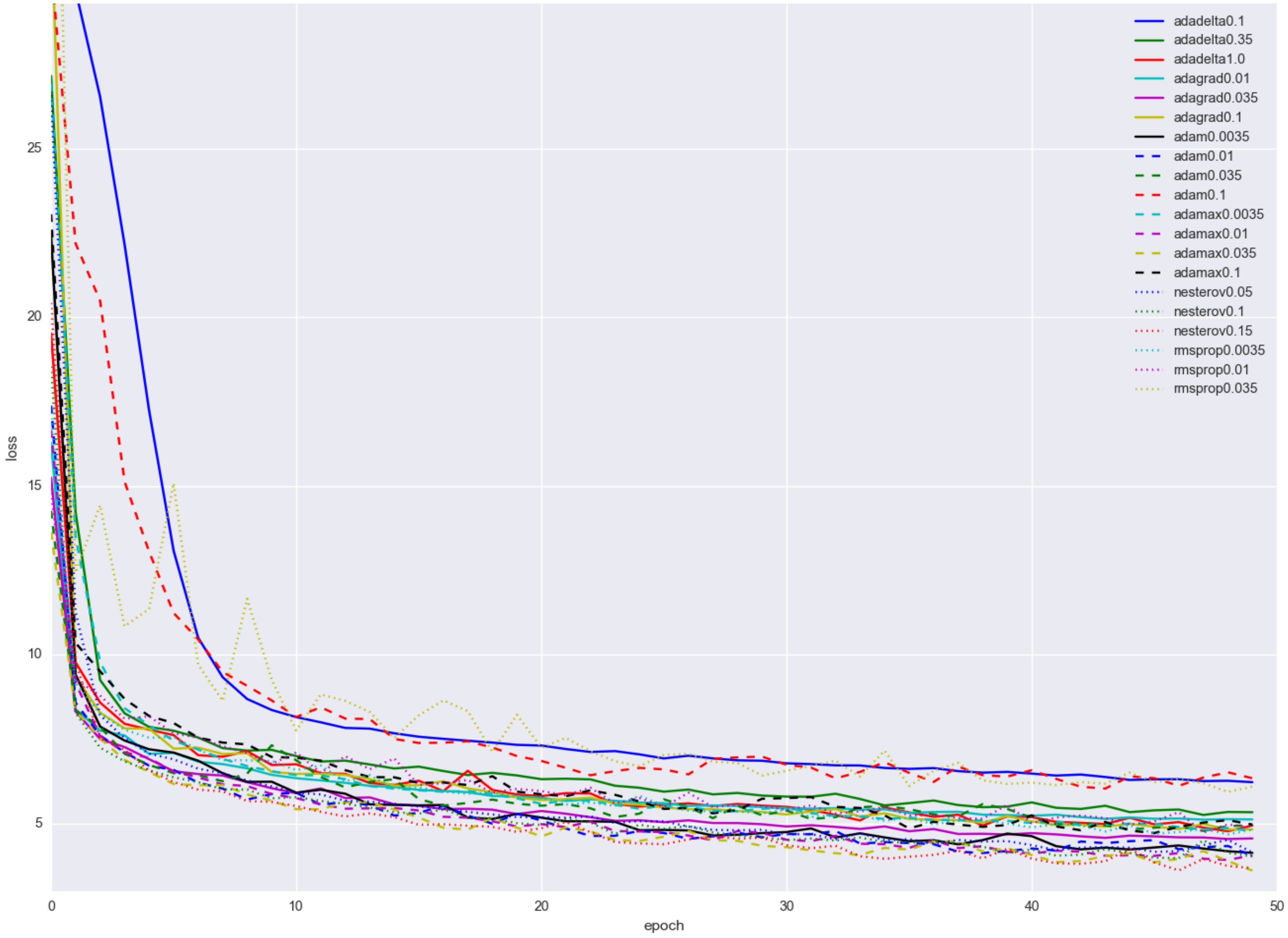
- Пока не сойдется значение Q и/или веса выполняется следующее
  - выбирается случайно объект x_i из X^l
  - вычисляется значение функции потери для этого объекта $\varepsilon_i := \mathscr{L}_i(w);$
  - вычисляется градиент $G_i := \nabla \mathscr{L}_i(w);$
  - делается градиентный шаг $w := w - h \sum_{i=1}^{\ell} G_i;$
  - оценивается функционал $\bar{Q} := (1 - \lambda)\bar{Q} + \lambda\varepsilon_i;$

Stochastic Average Gradient

Как это в оригинальной статье. d – переменная, отвечающая за накопление суммы.

Algorithm 1 Basic SAG method for minimizing $\frac{1}{n}\sum_{i=1}^{n} f_i(x)$ with step size α.

$d = 0$, $y_i = 0$ for $i = 1, 2, \ldots, n$
for $k = 0, 1, \ldots$ **do**
  Sample i from $\{1, 2, \ldots, n\}$
  $d = d - y_i + f_i'(x)$
  $y_i = f_i'(x)$
  $x = x - \frac{\alpha}{n} d$
end for



adadelta0.1
adadelta0.35
adadelta1.0
adagrad0.01
adagrad0.035
adagrad0.1
adam0.0035
adam0.01
adam0.035
adam0.1
adamax0.0035
adamax0.01
adamax0.035
adamax0.1
nesterov0.05
nesterov0.1
nesterov0.15
rmsprop0.0035
rmsprop0.01
rmsprop0.035
loss
25
20
15
10
5
0
10
20
30
40
50
epoch

Используемые источники

1. http://ruder.io/optimizing-gradient-descent/index.html#gradientdescentvariants
2. https://habrahabr.ru/post/318970/
3. https://arxiv.org/pdf/1309.2388.pdf
4. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf
5. https://en.wikipedia.org/wiki/Stochastic_gradient_descent
6. https://github.com/esokolov/ml-course-hse/blob/master/2017-fall/lecture-notes/lecture02-linregr.pdf
7. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/