DL: Optimization

torch.optim

Градиентный спуск

Градиентный спуск (GD)

Задача оптимизации:

$$L(w) = \frac{1}{n} \sum L(w, x_i, y_i) \rightarrow \min_w$$

Градиент указывает направление максимального роста

$$\nabla L(w) = \left(\frac{\partial L(w, x_i, y_i)}{\partial w_0}, \frac{\partial L(w, x_i, y_i)}{\partial w_1}, \frac{\partial L(w, x_i, y_i)}{\partial w_k} \right)$$

Делаем шаг в сторону противоположную направлению градиента

$$w_j = w_{j-1} - \eta \nabla L(w)$$

Сходимость

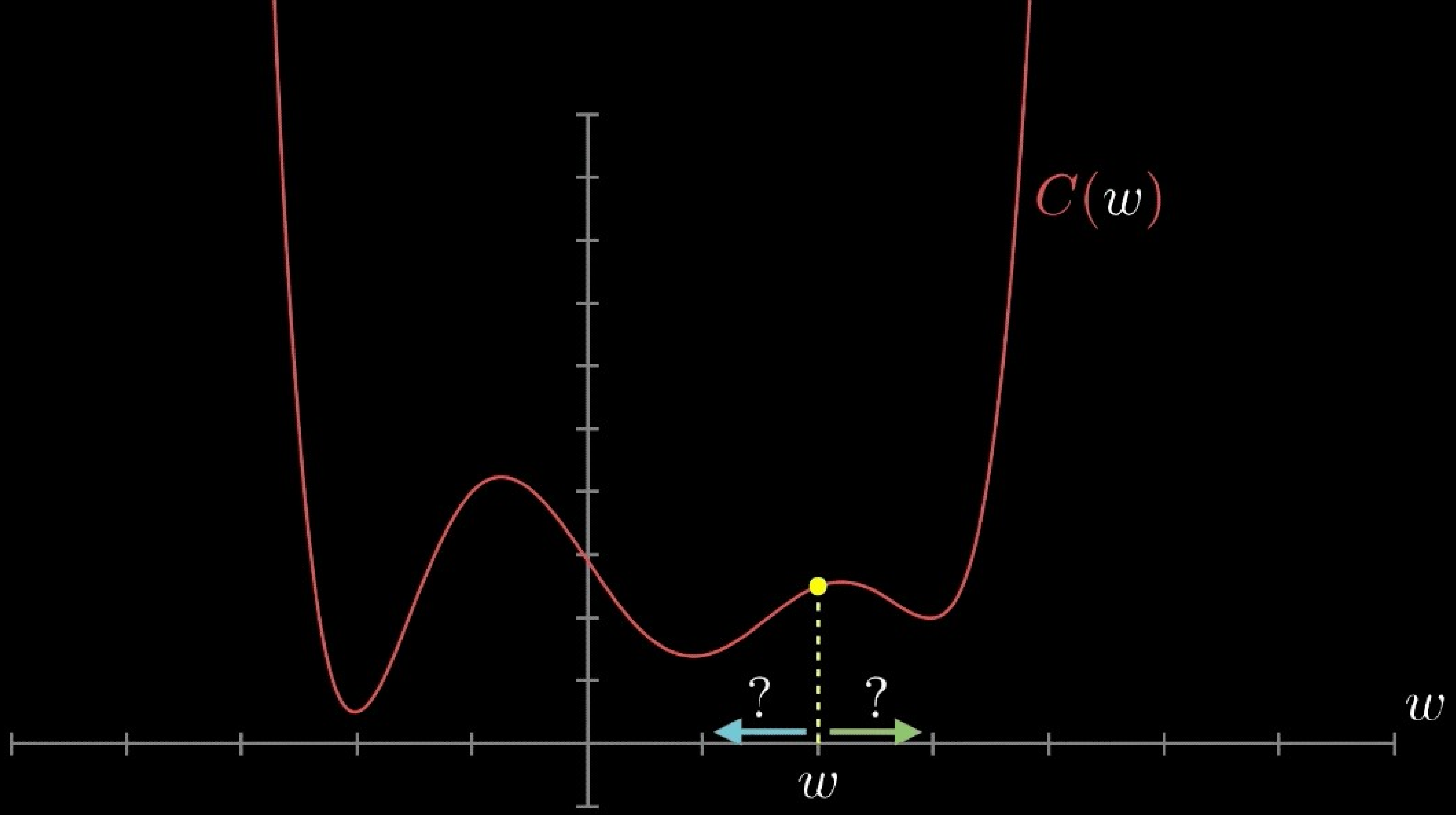
Случай 1:

$$\left\| w_j - w_{j-1} \right\| < \epsilon$$

Случай 2:

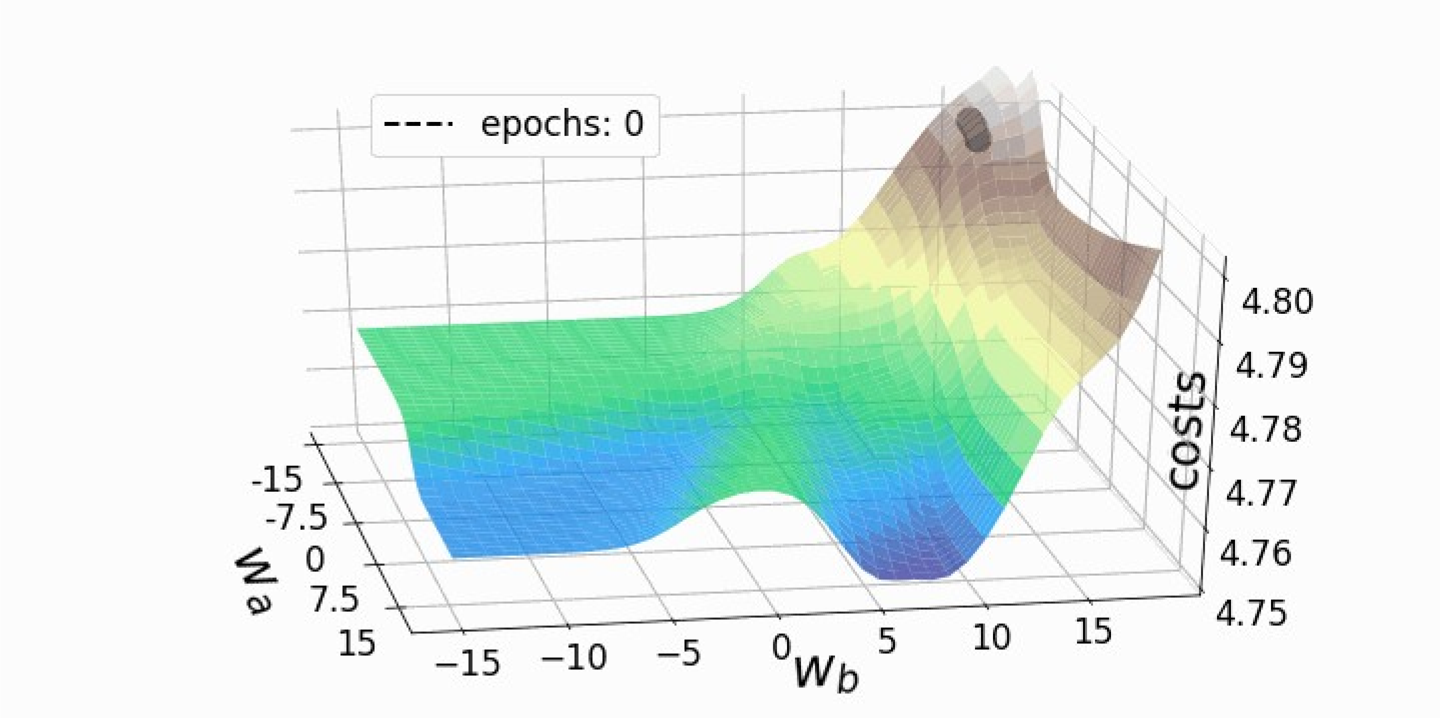
$$\left\| \nabla L(w) \right\| < \epsilon$$

Случай 3 (правило в глубоком обучении): останавливаем процесс, когда ошибка на виладиционном множестве перестает уменьшаться



$C(w)$
?
?
w
w

Градиентный спуск



Градиентный спуск (GD): линейная регрессия

Задача оптимизации:

$$L(w)=\frac{1}{n}\sum\left(y_i-x_i^T w\right)^2\rightarrow \min_w$$

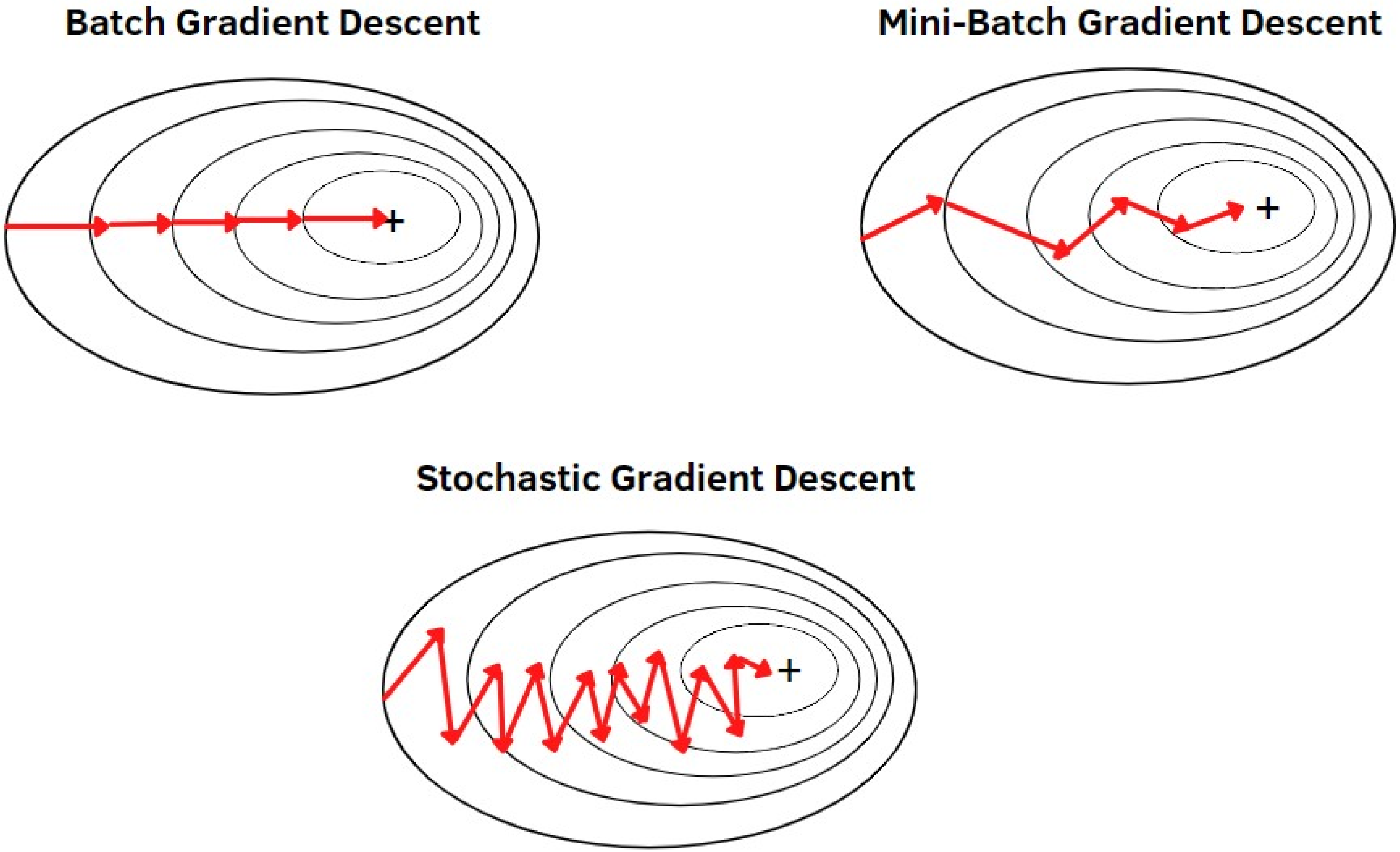
Градиент:

$$\nabla L(w)=\frac{-2}{n}\sum\left(y_i-x_i^T w\right)x_i$$

Делаем шаг в сторону противоположную направлению градиента

$$w_j=w_{j-1}+0.1\frac{2}{n}\sum\left(y_i-x_i^T w\right)x_i$$

GD vs SGD


Batch Gradient Descent
Mini-Batch Gradient Descent
Stochastic Gradient Descent

Стохастический градиентный спуск

- Оценка по одному объекту не смещенная
- Длина шага должна зависеть от номера итерации

$$\sum \eta_i = \infty \qquad \sum \eta_i^2 < \infty$$

- Сходимость к глобальному оптимуму гарантируется только для выпуклых функций

Стохастический градиентный спуск: используйте то, что работает

$$\eta_i = \frac{0.1}{i^{0.3}}$$

GD vs SGD

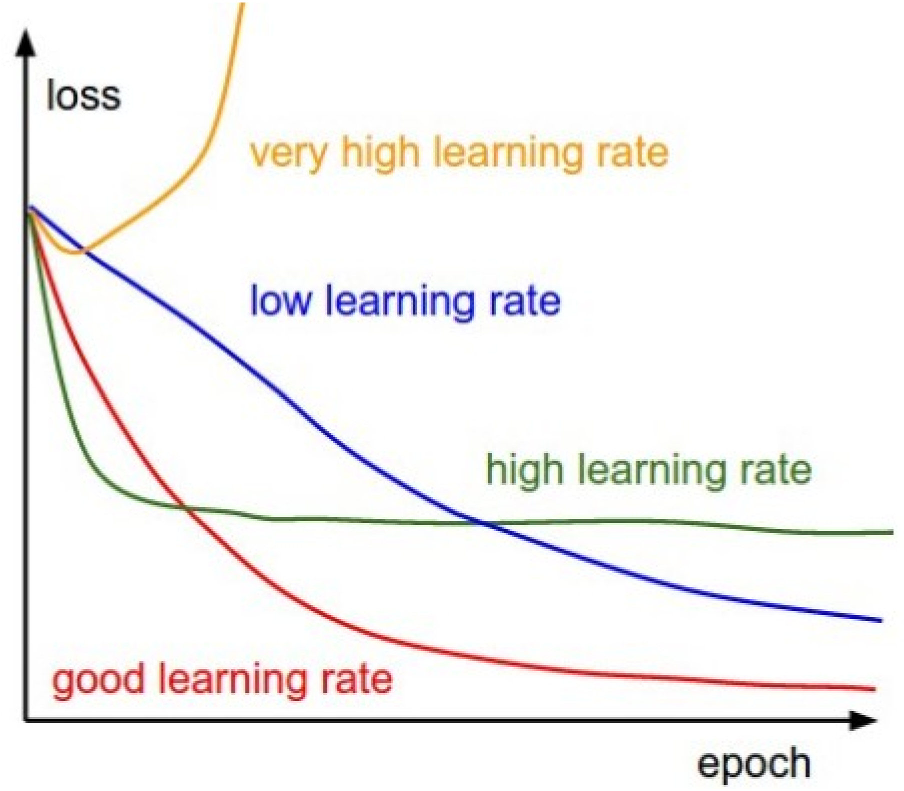
- И для GD и для SGD нет гарантий глобального минимума, сходимости
- SGD быстрее, на каждой итерации используется только одно наблюдение
- Для SGD спуск очень зашумлён
- GD: $O(n)$, SGD: $O(1)$
- Шум в оценке градиента помогает выпрыгивать из локальных оптимумов

Mini-bath SGD

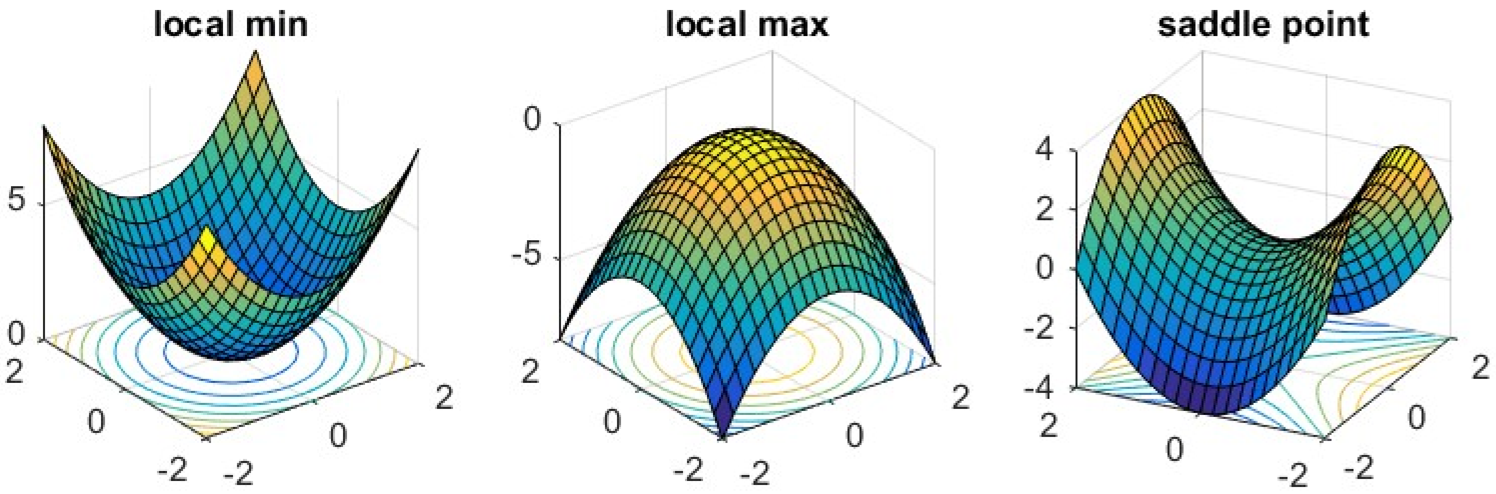
- Размер батча обычно десятки или сотни наблюдений
- Имеет смысл брать степень двойки
- Возможно, делает оценку градиента более стабильной
- За счёт векторизации более эффективен, чем шаг по одному объекту

Скорость обучения

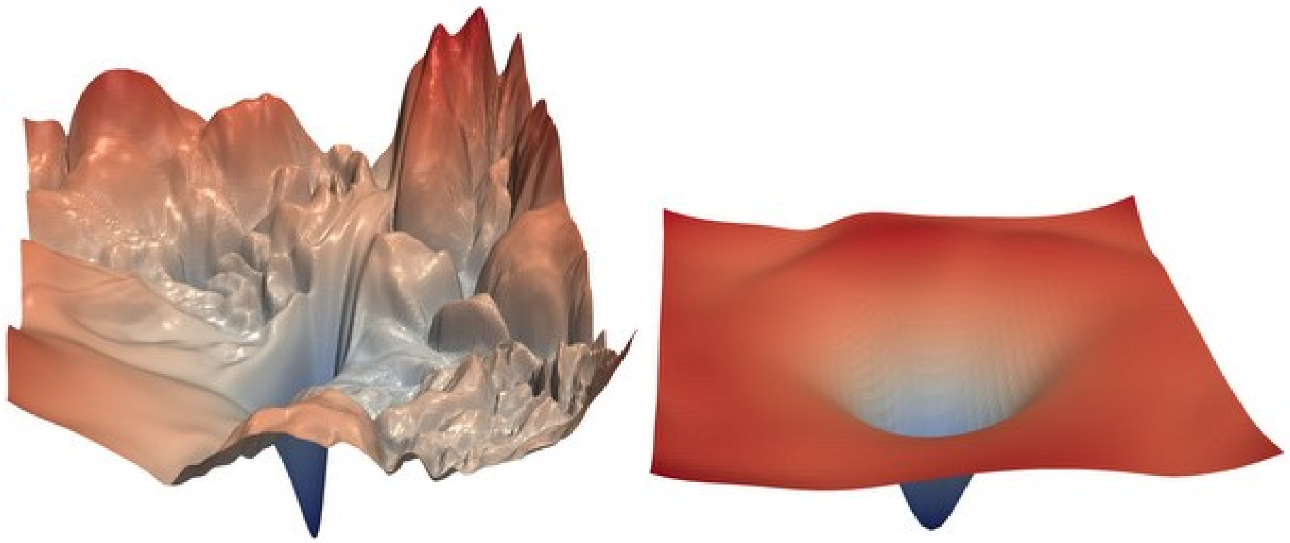
Скорость обучения η надо подбирать аккуратно, если она будет большой, мы можем скакать вокруг минимума, если маленькой — очень долго к нему двигаться



Седловые точки



Визуализация потерь



https://github.com/tomgoldstein/loss-landscape?tab=readme-ov-file

Проблемы SGD

К обновлению всех параметров применяется одна и та же скорость обучения. Возможно, что какие-то параметры приходят в оптимальную точку быстрее, и их не надо обновлять.

Momentum SGD

- Расчет SGD:

$$g_t = \frac{1}{n} \sum \nabla L(w_{t-1}, x_i, y_i) \qquad \Delta w_t = -\eta g_t$$

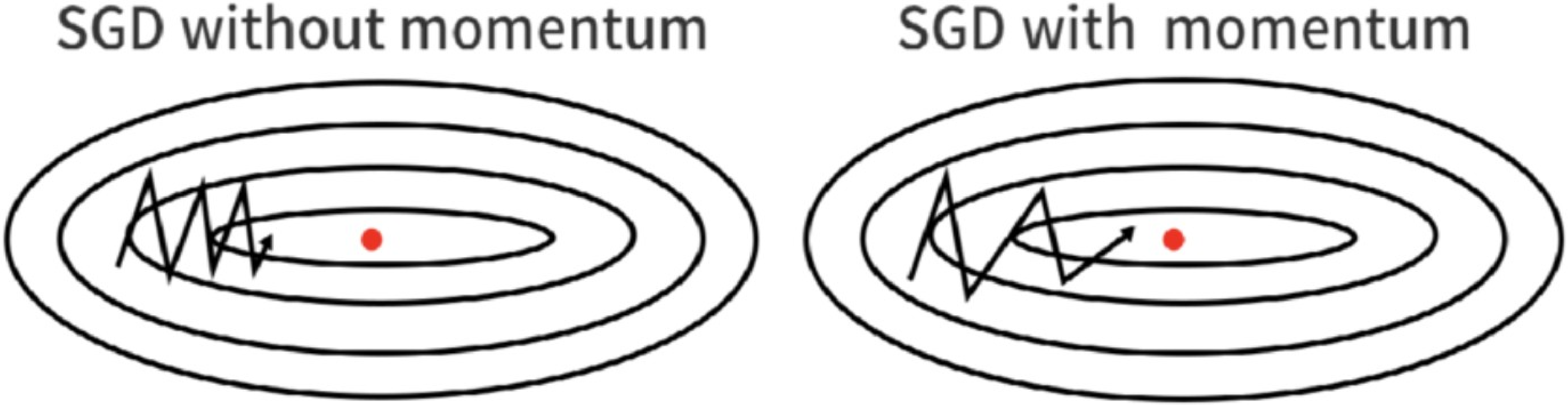
Идея

- Запомним направление и силу градиента на предыдущем шаге

$$u_t = \gamma u_{t-1} + \eta g_t$$

$$\Delta w_j = -u_t$$

Momentum SGD vs SGD



https://distill.pub/2017/momentum/

Адаптивные методы

- Заметим, что до сих пор скорость обучения была одна во всех направлениях, мы пытались выбрать направление как бы глобально
- Может сложиться, что некоторые веса уже близки к своим локальным минимумам, по этим координатам надо двигаться медленнее, а по другим быстрее $\Rightarrow$ адаптивные методы градиентного спуска
- Идея: давайте быстрее двигаться по тем параметрам, которые не сильно меняются, и медленнее по быстро меняющимся параметрам.

AdaGrad

$$G_t^j = G_{t-1}^j + g_{tj}^2 \qquad \Delta w_j = \frac{-\eta}{\sqrt{G_t^j + \epsilon}} g_t^j$$

- g_t^j градиент по j-му параметру
- для каждого параметра своя скорость обучения
- G_t^j всегда увеличивается

RMSprop

$$G_t^j = \alpha G_{t-1}^j + (1-\alpha) g_{tj}^2$$

$$\Delta w_j = \frac{-\eta}{\sqrt{G_t^j + \epsilon}} g_t^j$$

- Скорость обучения адаптируется к последнему сделанному шагу, бесконтрольного роста ? больше не происходит
- RMSprop нигде не был опубликован, Хинтон просто привёл его в своей лекции, сказав, что это норм тема

Adam (Adaptive Moment Estimation)

- Комбинируем Momentum и индивидуальные скорости обучения

$$u_t^j = \frac{\beta_1 u_{t-1}^j + (1-\beta_1) g_{tj}}{1-\beta_1}$$

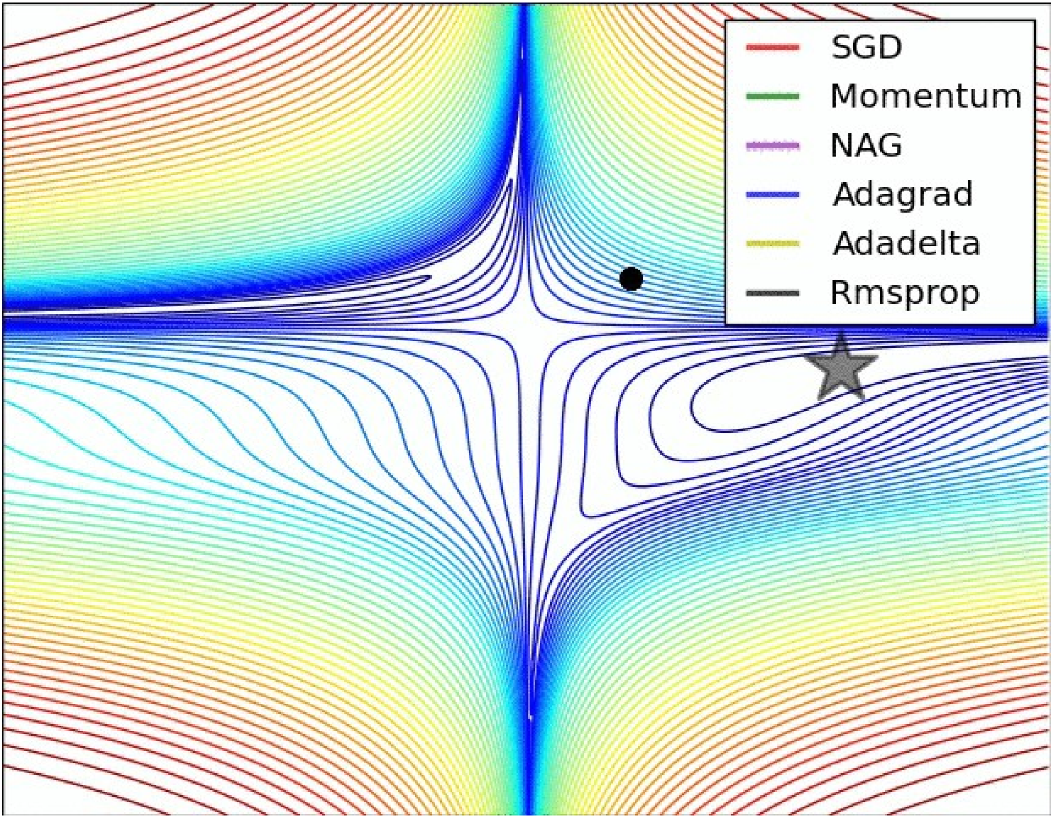
$$G_t^j = \frac{\beta_2 G_{t-1}^j + (1-\beta_2) g_{tj}^2}{1-\beta_2}$$

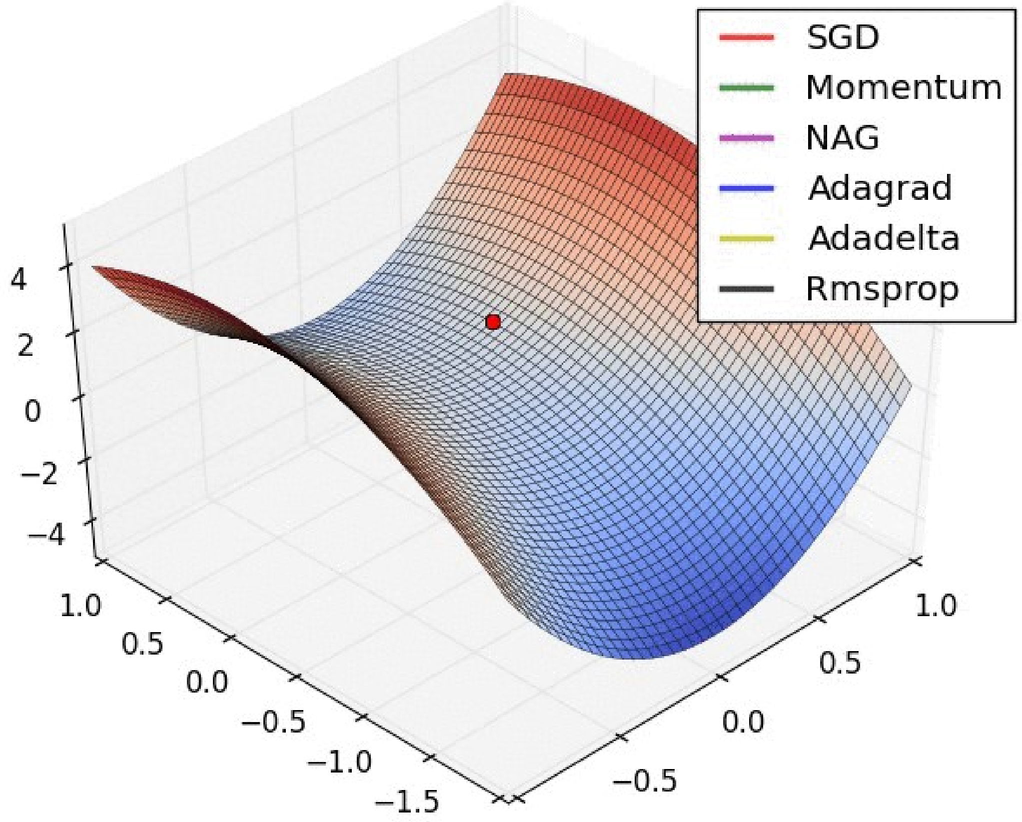
$$\Delta w_j = \frac{-\eta}{\sqrt{G_t^j + \epsilon}} u_t^j$$

Резюме

- Momentum SGD сохраняет направление шага и позволяет добиваться более быстрой сходимости
- Адаптивные методы позволяют находить индивидуальную скорость обучения для каждого параметра
- Adam комбинирует в себе оба подхода

Резюме


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

Нейросети и кросс-валидация

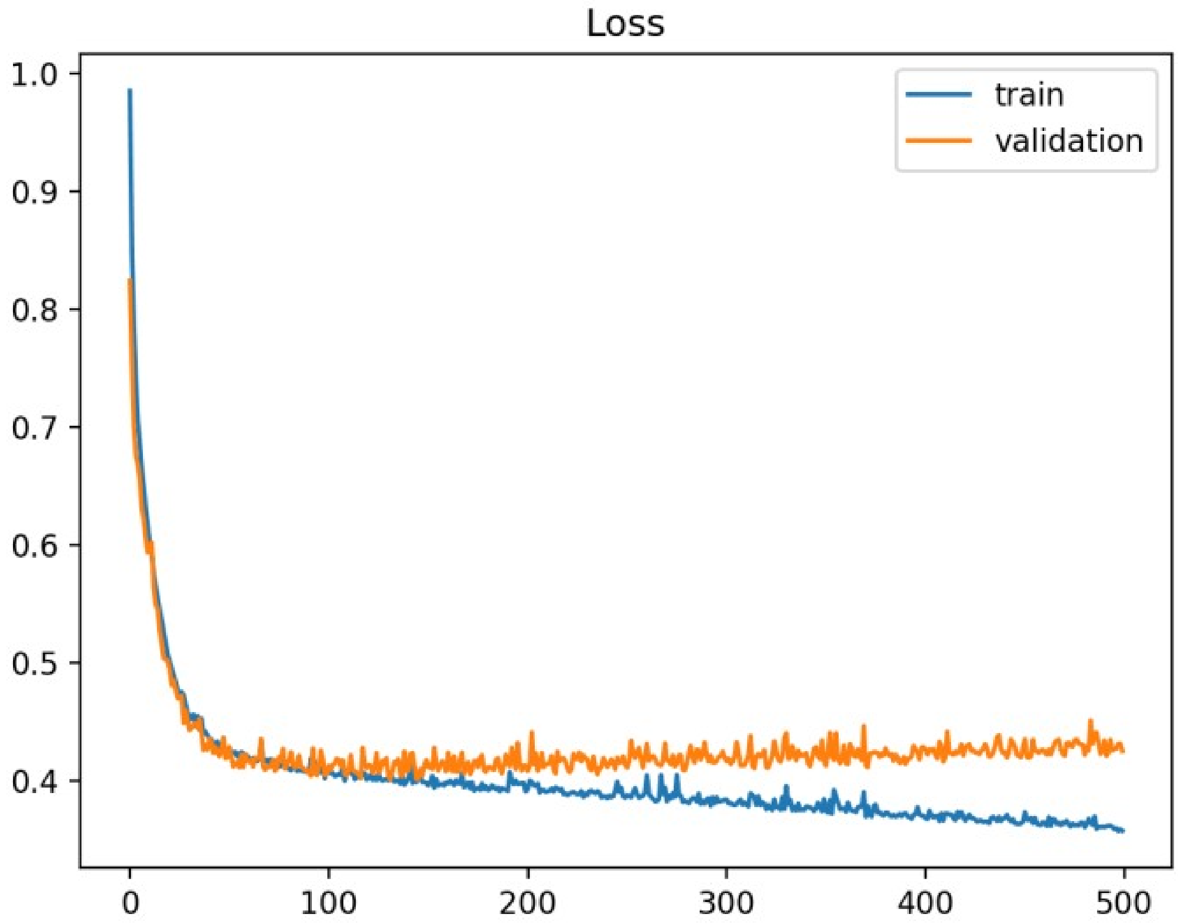
- Кросс-валидацию для нейронных сетей обычно не делают
- Сетка учится долго, дробить выборку на части и обучать несколько экземпляров очень дорого
- Перебор гиперпараметров по решётке обычно не делают, так как это тоже дорого
- При экспериментах делают одно какое-то изменение за раз и запускают обучение

Эпохи и батчи

- Эпоха — один проход по данным
- Батч (пакет) часть данных, которая в данный момент участвует в обучении
- Данные перед каждой эпохой желательно перемешивать (shuffle)

Контроль обучения

- Значение функции потерь в зависимости от итерации (проверка идёт ли оптимизация)
- Обучающая выборка не покажет переобучение ⇒ следим за ошибкой на валидационной выборке
- ВАЖНО: использовать валидацию, а не тест
- Число эпох для обучения - гиперпараметр, на валидации можно понять, когда надо остановить обучение
- На тестовой выборке оцениваем окончательное качество модели


Loss
train
validation
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
100
200
300
400
500