

# Нейронные сети и их практическое применение.

Лекция 5. Обзор алгоритмов обучения.

Дмитрий Буряк  
к.ф.-м.н  
dyb04@yandex.ru

# Градиентные методы обучения

Минимизация целевой функции :

$d$  - желаемый выход

$y$  - реальный выход

$$E(w) = \frac{1}{2} \sum_j (y_j - d_j)^2$$

Метод обучения - градиентный спуск.

$$E(w + p) = E(w) + [g(w)]^T p + \frac{1}{2} p^T H(w) p + \dots$$

$$g(w) = \nabla E = \left[ \frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \dots, \frac{\partial E}{\partial w_n} \right]$$

$H(w)$ -матрица вторых производных;

$w$  - минимум  $E(w)$ , если  $g(w)=0$ ,  $H(w)$  – положительно определена

Обозначим  $w_k$  -решение, полученное на  $k$ -ом шаге.

Цель: подобрать  $p$  и  $\eta$  так, чтобы, для очередной точки  $w_{k+1}$  выполнялось

условие:

$$w_{k+1} = w_k + \eta_k p_k$$

$$E(w_{k+1}) < E(w_k)$$

# Схема универсального алгоритма обучения

1. Проверка оптимальности текущего решения  $w_k$ .
2. Определение вектора направления оптимизации  $p_k$  для точки  $w_k$ .
3. Выбор шага  $\eta_k$  в направлении  $p_k$ , при котором выполняется условие
$$E(w_{k+1}) < E(w_k)$$
4. Определение нового решения  $w_{k+1} = w_k + \eta_k p_k$

а также соответствующих ему значений функции ошибки, градиента, матрицы вторых производных.

Переход на 1.

# Алгоритм наискорейшего спуска

Линейное приближение функции  $E(w)$ .

Для выполнения условия  $E(w_{k+1}) < E(w_k)$ , надо  $g(w_k)^T p < 0$

$$p_k = -g(w_k)$$

Достоинства:

- небольшая вычислительная сложность;
- невысокие требования к памяти.

Недостатки:

- не учитывает информацию о кривизне функции;
- замедление в случае маленького градиента;
- медленная сходимость.

# Наискорейший спуск с моментом

$$\Delta w_k = \eta p_k + \alpha(w_k - w_{k-1})$$

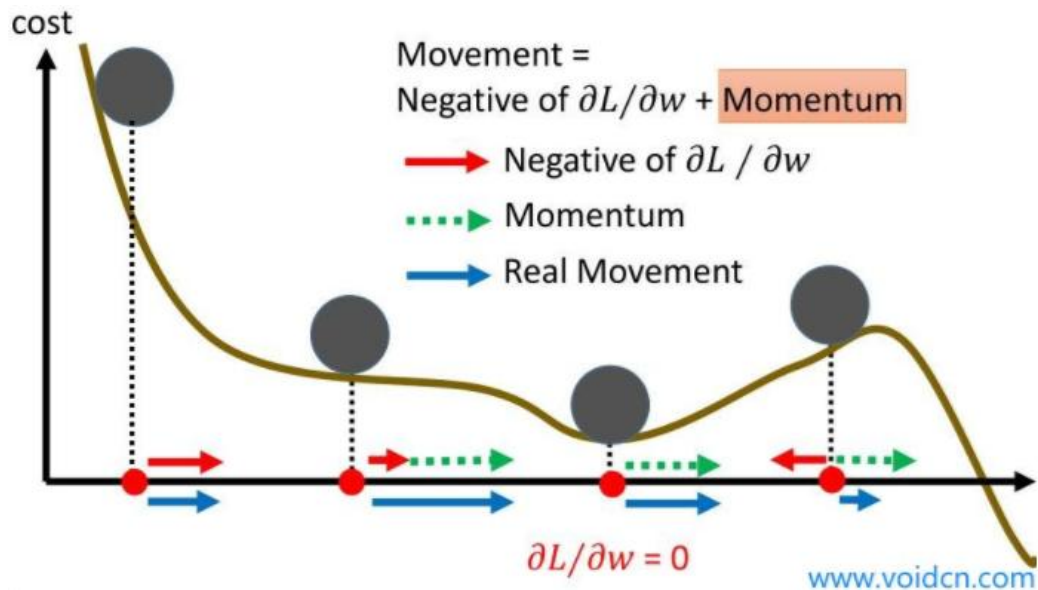
$\alpha$  - коэффициент момента в интервале [0,1]

На плоских участках:  $\Delta w_k = \eta p_k + \alpha \Delta w_k$

$$\Delta w_k = \frac{\eta}{1-\alpha} p_k$$

**Достоинство: ускорение сходимости на плоских участках, позволяет избежать локальных экстремумов.**

# Использование момента



Сделаем рекуррентные подстановки.

Введем ряд по  $t$ .

Решим разностное уравнение.

$$\Delta w_{ji}(n) = \eta \sum_{t=0}^n \alpha^{n-t} \delta_j(t) y_i(t).$$

$$\delta_j(n) y_i(n) = -\partial \mathbf{E}(n) / \partial w_{ji}(n)$$

$$\Delta w_{ji}(n) = -\eta \sum_{t=0}^n \alpha^{n-t} \frac{\partial \mathbf{E}(t)}{\partial w_{ji}(t)}$$

1. Для сходимости ряда необходимо  $0 \leq |\alpha| < 1$
2. Ускоряет спуск, если знак градиента не меняется
3. Замедляет спуск, если знак градиента изменяется – стабилизирующий эффект.

# Метод переменной метрики

Квадратичное приближение  $E(w)$

$$E(w_k + p_k) = E(w_k) + [g(w_k)]^T p_k + \frac{1}{2} p_k^T H(w_k) p_k$$

$$\frac{dE(w_k + p_k)}{dp_k} = 0 \quad g(w_k) + H(w_k) p_k = 0$$

$$p_k = -[H(w_k)]^{-1} g(w_k)$$

Используют приближение  $H(w)$ :  $V_k = V_{k-1} + F(w_k, w_{k-1}, g(w_k), g(w_{k-1}))$

где:  $V_k$  - приближение  $[H(w_k)]^{-1}$

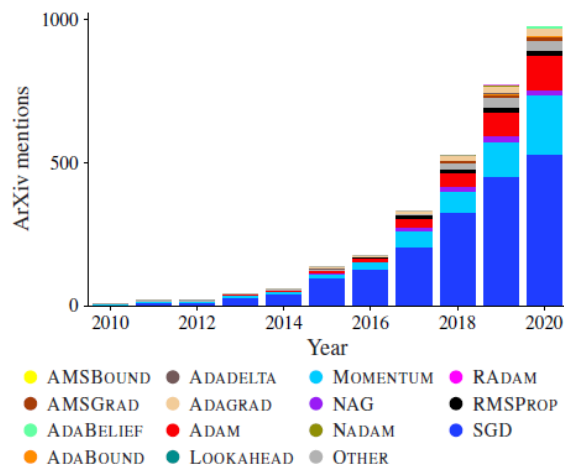
Достоинство: быстрая сходимость.

Недостатки:

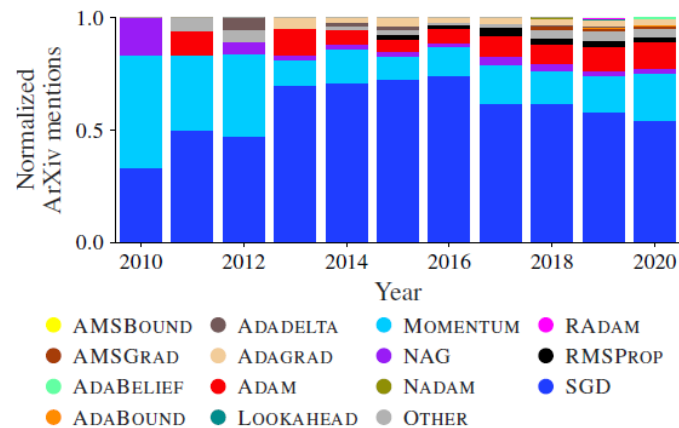
- вычислительная сложность;
- требования к памяти.

# Выбор алгоритма обучения

- ❑ Существует более 100 алгоритмов оптимизации для НС\*.
- ❑ Эмпирическое сравнение эффективности различных алгоритмов.
- ❑ Не существует исследований, которые сравнивают все (почти все) алгоритмы.



**Количество упоминаний оптимизаторов в статьях на сайте arxiv.org\***



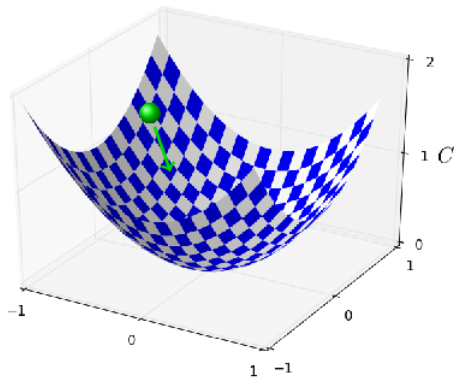
**Относительное число упоминаний оптимизаторов в статьях на сайте arxiv.org\***

\* <https://arxiv.org/abs/2007.01547>



# Стохастический градиентный спуск (SGD)

- ❑ Основной алгоритм для глубокого обучения.
- ❑ Базируется на принципах градиентного спуска.
- ❑ Вычисление градиента по каждому примеру вычислительно затратно → выбор случайных подмножеств из обучающей выборки.
- ❑ SGD с моментом



$$w_k \rightarrow w'_k = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_l \rightarrow b'_l = b_l - \eta \frac{\partial C}{\partial b_l}$$

$$\nabla C \approx \frac{1}{m} \sum_{j=1}^m \nabla C_{X_j}$$

$$w_k \rightarrow w'_k = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

$$b_l \rightarrow b'_l = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l}$$

# Алгоритм обучения Adam.

□ Момент + учет частоты  
обновления веса.

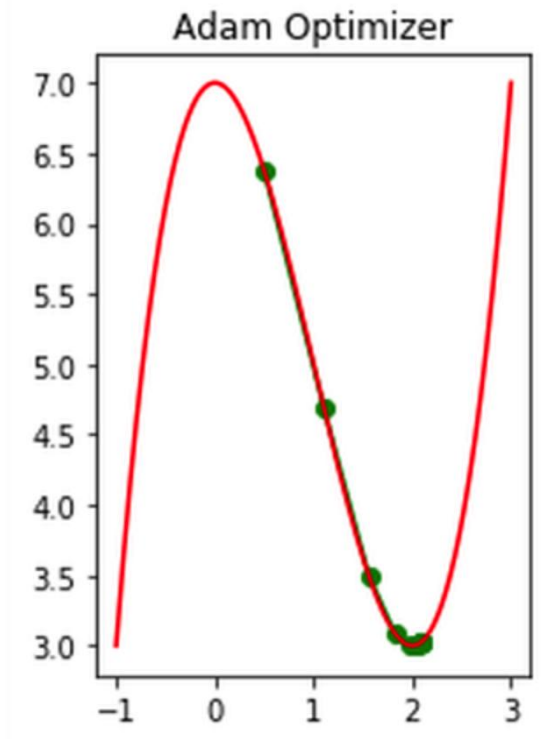
$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

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

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

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

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



# Оптимизаторы в Pytorch.

Пакет `torch.optim` -- коллекция различных оптимизаторов градиентного спуска, от простейшего SGD до современных:

- `optim.ASGD`,
- `optim.Adadelta`,
- `optim.Adagrad`,
- `optim.RMSprop`,
- `optim.Adam`

```
torch.manual_seed(42)
a = torch.randn(1, requires_grad=True, dtype=torch.float)
b = torch.randn(1, requires_grad=True, dtype=torch.float)
print("Initial values:", a, b)
lr = 1e-1
n_epochs = 1000
# Defines a SGD optimizer to update the parameters
optimizer = optim.SGD([a, b], lr=lr)

for epoch in range(n_epochs):
    yhat = a + b * x_train
    error = y_train - yhat
    loss = (error ** 2).mean()

    loss.backward()

    # a -= lr * a.grad
    # b -= lr * b.grad
    optimizer.step()
    optimizer.zero_grad()
```

# Методы оптимизации 2-го порядка

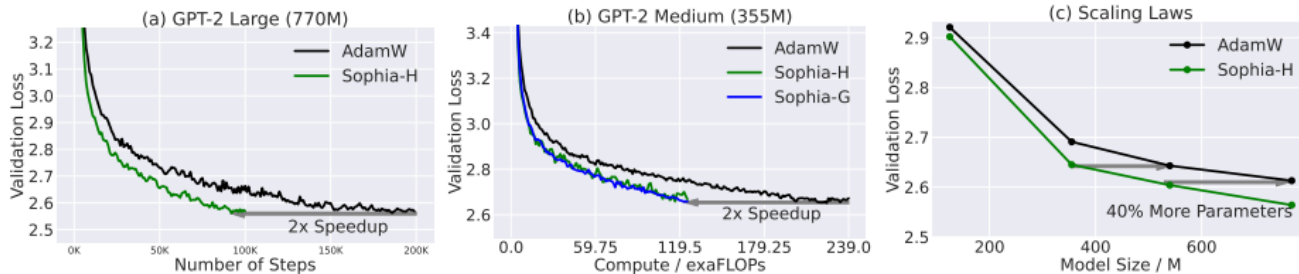
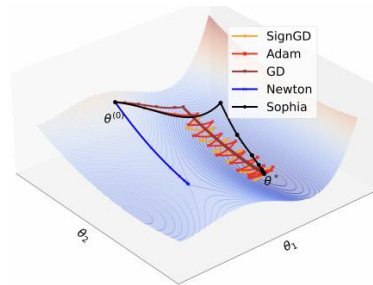
$$C(w + \Delta w) \approx C(w) + \nabla C \cdot \Delta w + \frac{1}{2} \Delta w^T H \Delta w.$$

$$\Delta w = -H^{-1} \nabla C.$$

- ❑ Оценка Гессиана  $\sim$  скорость изменения градиента  $\rightarrow$  обеспечение одинаковой сходимости по всем переменным
- ❑ Требуется много памяти: 1000000 весов  $\rightarrow 10^{12}$  элементов Гессиана  $\rightarrow >3\text{Tb}$  RAM .
- ❑ Методы частичной аппроксимации Гессиана (L-BFGS).
- ❑ Аппроксимация требует использования всех примеров из обучающей выборки.

# Sophia – оптимизатор 2-го порядка

- ❑ Усреднение градиента (скользящее среднее)
- ❑ Аппроксимация диагональных элементов Гессиана (2 варианта).
- ❑ Аппроксимация выполняется каждые  $n$  эпох.
- ❑ Применение клипирования для избегания слишком резких изменений весов.



- ❑ Представлен, как эффективный алгоритм обучения LLM: 2x меньше вычислений для того же минимума чем Adam, достижения такой же эффективности на моделях меньшего размера.

## Algorithm 3 Sophia

- 1: **Input:**  $\theta_1$ , learning rate  $\{\eta_t\}_{t=1}^T$ , hyperparameters  $\lambda, \beta_1, \beta_2, \epsilon$ , and estimator choice Estimator  $\in \{\text{Hutchinson, Gauss-Newton-Bartlett}\}$
- 2: Set  $m_0 = 0, v_0 = 0, h_{1-k} = 0$
- 3: **for**  $t = 1$  to  $T$  **do**
- 4:   Compute minibatch loss  $L_t(\theta_t)$ .
- 5:   Compute  $g_t = \nabla L_t(\theta_t)$ .
- 6:    $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
- 7:   **if**  $t \bmod k = 1$  **then**
- 8:     Compute  $\hat{h}_t = \text{Estimator}(\theta_t)$ .
- 9:      $h_t = \beta_2 h_{t-k} + (1 - \beta_2) \hat{h}_t$
- 10:   **else**
- 11:      $h_t = h_{t-1}$
- 12:      $\theta_t = \theta_t - \eta_t \lambda \theta_t$  (weight decay)
- 13:      $\theta_{t+1} = \theta_t - \eta_t \cdot \text{clip}(m_t / \max\{h_t, \epsilon\}, \rho)$

# Изменение скорости обучения

## ❑ Ступенчатое уменьшение

- в  $k$  раз каждые  $n$  итераций обучения;
- уменьшение коэффициента обучения, если ошибка на подтверждающей выборке перестала уменьшаться.

## ❑ Экспоненциальное $\eta = \alpha_0 e^{-\beta t}$

## ❑ Гиперболическое $\eta = \frac{\alpha_0}{1 + \beta t}$

# Схемы изменения скорости обучения

Name	Ref.	Illustration
Constant		
Step Decay	constant factor	
	multi-step	
Smooth Decay	linear decay	e.g. (Goodfellow et al., 2016)
	polynomial decay	
	exponential decay	
	inverse time decay	e.g. (Bottou, 2012)
	cosine decay	(Loshchilov & Hutter, 2017)
	linear cosine decay	(Bello et al., 2017)

\* <https://arxiv.org/abs/2007.01547>

# Стохастический метод обучения.

## Общая схема

1. Выполнить начальную инициализацию весов.
2. Вычислить выход для примера из обучающей выборки.
3. Вычислить ошибку:
4. Выбрать случайным образом вес и изменить его значение на случайную величину.  
Если проведенное изменение уменьшает целевую функцию (ошибку), то сохранить его, иначе вернуться к прежнему значению веса.
5. Повторять шаги 2, 3, 4, пока сеть не обучится.



# Алгоритм имитации отжига

1. Выполнить начальную инициализацию весов. Задать начальную температуру  $T=T_{max}$ . Вычислить значение ошибки.

2. Пока  $T>0$  повторить  $L$  раз действия:

2.1. Выполнить случайную коррекцию весов  $w'=w+\Delta w$ .

2.2. Вычислить разницу целевых функций  $c=E(w')-E(w)$ .

2.3. Если  $c<0$ , то  $w=w'$

иначе принять  $w=w'$  с вероятностью

$$P(c) = e^{-\frac{c}{kT}}$$

3. Уменьшить температуру  $T$ .

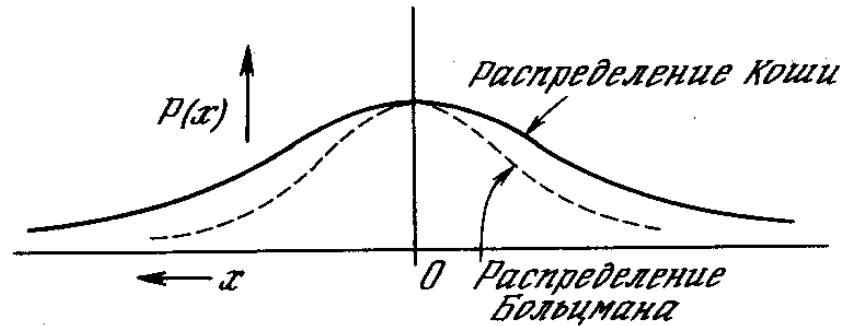
4. Повторять 2, 3, 4 пока  $T>0$  или значение ошибки больше порога.

Приращение весов  $\Delta w$ : 
$$P(\Delta w) = e^{-\frac{\Delta w^2}{T^2}}$$

# Алгоритм имитации отжига (Коши)

Скорость сходимости метода Больцмана:  $T(t) = \frac{T_{\max}}{\log(1+t)}$

Распределение Коши:  $P(x) = \frac{T(t)}{T(t)^2 + x^2}$



Скорость сходимости метода Коши:  $T(t) = \frac{T_{\max}}{1+t}$

# Комбинированный метод обучения

Объединение традиционного обратного распространения с обучением Коши.

$$\Delta w_k = \beta[\eta_k p_k + \alpha(w_k - w_{k-1})] + (1 - \beta)\Delta w^c$$

# Вопросы

1. Основные достоинства и недостатки алгоритма наискорейшего спуска?
2. Что препятствует применению алгоритмов, основанных на методах оптимизации второго порядка, для обучения современных нейронных сетей?
3. Какое основное преимущество стохастических алгоритмов обучения перед градиентными методами?