

# Введение в искусственный интеллект. Современное компьютерное зрение

## Лекция 4. Обучение нейронных сетей

Бабин Д.Н., Иванов И.Е., Петюшко А.А.

кафедра Математической Теории Интеллектуальных Систем

16 марта 2021 г.



- 1 Инициализация весов нейронной сети
- 2 Градиентный спуск
- 3 Модификации градиентного спуска для обучения нейронных сетей
- 4 Граф вычислений
- 5 Метод обратного распространения ошибки



## 1 Выбор архитектуры



# Этапы подготовки модели к обучению

- 1 Выбор архитектуры
- 2 Выбор функции потерь



# Этапы подготовки модели к обучению

- 1 Выбор архитектуры
- 2 Выбор функции потерь
- 3 **Инициализация весов**



# Этапы подготовки модели к обучению

- 1 Выбор архитектуры
- 2 Выбор функции потерь
- 3 **Инициализация весов**
- 4 Решение оптимизационной задачи



- Инициализация нулями



# Методы инициализации весов нейронной сети

- Инициализация нулями
- Инициализация единицами или другими константами





# Методы инициализации весов нейронной сети

- Инициализация нулями
- Инициализация единицами или другими константами
- Инициализация нормально распределенными значениями



# Методы инициализации весов нейронной сети

- Инициализация нулями
- Инициализация единицами или другими константами
- Инициализация нормально распределенными значениями
- Инициализация равномерно распределенными значениями



# Методы инициализации весов нейронной сети

- Инициализация нулями
- Инициализация единицами или другими константами
- Инициализация нормально распределенными значениями
- Инициализация равномерно распределенными значениями
- Инициализация значениями из усеченного нормального распределения



# Методы инициализации весов нейронной сети

- Инициализация нулями
- Инициализация единицами или другими константами
- Инициализация нормально распределенными значениями
- Инициализация равномерно распределенными значениями
- Инициализация значениями из усеченного нормального распределения
- Инициализация случайной ортогональной матрицей



# Методы инициализации весов нейронной сети

- Инициализация нулями
- Инициализация единицами или другими константами
- Инициализация нормально распределенными значениями
- Инициализация равномерно распределенными значениями
- Инициализация значениями из усеченного нормального распределения
- Инициализация случайной ортогональной матрицей
- Инициализация единичной матрицей



## Пример многослойной сети

$$y = W_{10} W_9 W_8 W_7 \dots W_3 W_2 W_1 x$$



## Пример многослойной сети

$$y = W_{10} W_9 W_8 W_7 \dots W_3 W_2 W_1 x$$

## Взрывающийся градиент (exploding gradient)

Если  $W_i = \text{diag}(1.5, \dots, 1.5)$ , то  $y_i = (1.5)^i x$  и

$$y = (1.5)^{10} x$$



# Проблемы обучения нейронных сетей

## Пример многослойной сети

$$y = W_{10} W_9 W_8 W_7 \dots W_3 W_2 W_1 x$$

## Взрывающийся градиент (exploding gradient)

Если  $W_i = \text{diag}(1.5, \dots, 1.5)$ , то  $y_i = (1.5)^i x$  и

$$y = (1.5)^{10} x$$

## Затухающий градиент (vanishing gradient)

Если  $W_i = \text{diag}(0.5, \dots, 0.5)$ , то  $y_i = (0.5)^i x$  и

$$y = (0.5)^{10} x$$



## Вопросы

- Если инициализировать веса вероятностным распределением, то какие параметры оно должно иметь?
- Что мы хотим получить от инициализации?



## Вопросы

- Если инициализировать веса вероятностным распределением, то какие параметры оно должно иметь?
- Что мы хотим получить от инициализации?

## Предположения

- Веса независимы и одинаково распределены с нулевым средним
- Входы независимы и одинаково распределены с нулевым средним
- Веса и входы независимы



## Лемма

Для независимых случайных величин  $X$  и  $Y$  выполнено

$$D(XY) = E[X]^2 D(Y) + E[Y]^2 D(X) + D(X)D(Y)$$

$$D(w_i x_i) = E[x_i]^2 D(w_i) + E[w_i]^2 D(x_i) + D(w_i)D(x_i)$$

$$D(y_j) = D\left(\sum_{i=0}^{n_{in}} w_{ji} x_i\right) = n_{in} D(w_{ji}) D(x_i)$$



## Лемма

Для независимых случайных величин  $X$  и  $Y$  выполнено

$$D(XY) = E[X]^2 D(Y) + E[Y]^2 D(X) + D(X)D(Y)$$

$$D(w_i x_i) = E[x_i]^2 D(w_i) + E[w_i]^2 D(x_i) + D(w_i)D(x_i)$$

$$D(y_j) = D\left(\sum_{i=0}^{n_{in}} w_{ji} x_i\right) = n_{in} D(w_{ji}) D(x_i)$$

## Заключение

Для того, чтобы было выполнено, что  $D(y) = D(x)$ , веса необходимо инициализировать с

$$D(w) = \frac{1}{n_{in}}$$

# Пример подсчета дисперсии

## Пример

Рассмотрим, что происходит, если  $w_i \sim U[-\frac{1}{\sqrt{n_{in}}}, \frac{1}{\sqrt{n_{in}}}]$ .



## Пример

Рассмотрим, что происходит, если  $w_i \sim U[-\frac{1}{\sqrt{n_{in}}}, \frac{1}{\sqrt{n_{in}}}]$ .

$$D(w) = \frac{1}{12} \left( \frac{1}{\sqrt{n_{in}}} + \frac{1}{\sqrt{n_{in}}} \right)^2 = \frac{1}{3n_{in}}$$



# Пример подсчета дисперсии

## Пример

Рассмотрим, что происходит, если  $w_i \sim U[-\frac{1}{\sqrt{n_{in}}}, \frac{1}{\sqrt{n_{in}}}]$ .

$$D(w) = \frac{1}{12} \left( \frac{1}{\sqrt{n_{in}}} + \frac{1}{\sqrt{n_{in}}} \right)^2 = \frac{1}{3n_{in}}$$

То есть получаем, что  $D(y) = \frac{1}{3}D(x)$ , то есть сигнал затухает.





# Более продвинутые методы инициализации весов нейронной сети

- Лесюн-инициализация <sup>1</sup> ( $n_i$  — количество входов (нейронов)  $i$ -го слоя)

$$W \sim U[-\sqrt{\frac{3}{n_i}}, \sqrt{\frac{3}{n_i}}], \quad b \sim 0$$

---

<sup>1</sup>LeCun Y. A. et al. Efficient backprop. 1998.

<sup>2</sup>**Xavier** Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks.   2010.



# Более продвинутые методы инициализации весов нейронной сети

- Лесун-инициализация <sup>1</sup> ( $n_i$  — количество входов (нейронов)  $i$ -го слоя)



$$W \sim U[-\sqrt{\frac{3}{n_i}}, \sqrt{\frac{3}{n_i}}], \quad b \sim 0$$

- Xavier-инициализация <sup>2</sup>

$$W \sim U[-\frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}, \frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}], \quad b \sim 0$$

---

<sup>1</sup>LeCun Y. A. et al. Efficient backprop. 1998.

<sup>2</sup>**Xavier** Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks.   2010.

# Подсчет дисперсии для ReLU сетей

Продолжим выкладки:

$$\begin{aligned} D(y_i) &= n_{in}(E[w_{ij}]^2 D(w_i) + E[w_{ij}]^2 D(x_j) + D(w_{ij})D(x_j)) = \\ &= n_{in}(E[x_j]^2 D(w_{ij}) + 0 \cdot D(x_j) + D(w_{ij})D(x_j)) = \\ &= n_{in}(E[x_j]^2 D(w_{ij}) + D(w_{ij})D(x_j)) = n_{in}D(w_{ij})E[x_j^2] \end{aligned}$$



# Подсчет дисперсии для ReLU сетей

Продолжим выкладки:

$$\begin{aligned} D(y_i) &= n_{in}(E[w_{ij}]^2 D(w_i) + E[w_{ij}]^2 D(x_j) + D(w_{ij})D(x_j)) = \\ &= n_{in}(E[x_j]^2 D(w_{ij}) + 0 \cdot D(x_j) + D(w_{ij})D(x_j)) = \\ &= n_{in}(E[x_j]^2 D(w_{ij}) + D(w_{ij})D(x_j)) = n_{in}D(w_{ij})E[x_j^2] \end{aligned}$$

Подсчитаем как меняется дисперсия при использовании функции активации ReLU

$$E[x^2] = \int_{-\infty}^{+\infty} x^2 p(x) dx = \int_{-\infty}^{+\infty} \max(0, t)^2 p(t) dt = \int_0^{+\infty} t^2 p(t) dt = 0.5 \int_{-\infty}^{+\infty} t^2 p(t) dt = 0.5 D(t)$$

Получаем, что  $D(y_i) = n_{in}D(w_{ij})0.5D(t)$



- Инициализация для ReLU сетей <sup>3</sup>

$$W \sim N(0, \sqrt{\frac{2}{n_i}}), \quad b \sim 0$$

---

<sup>3</sup>K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. 2015.

- Инициализация предобученными моделями



# Инициализация предобученными моделями

- Инициализация предобученными моделями
- Инициализация предобученной без учителя моделью (автоэнкодеры)



# Этапы подготовки модели к обучению

- 1 Выбор архитектуры
- 2 Выбор функции потерь
- 3 Инициализация весов
- 4 **Решение оптимизационной задачи**

## Оптимизационная задача

$$L(x, \theta) = \sum_{(x_i, y_i) \in \text{train}} L(x_i, y_i, \theta) \rightarrow \min_{\theta}$$





## Оптимизационная задача

$$L(x, \theta) = \sum_{(x_i, y_i) \in \text{train}} L(x_i, y_i, \theta) \rightarrow \min_{\theta}$$

## Градиентный спуск

$$\omega[t+1] = \omega[t] - \alpha[t] \nabla (L, w[t])$$



## Оптимизационная задача

$$L(x, \theta) = \sum_{(x_i, y_i) \in \text{train}} L(x_i, y_i, \theta) \rightarrow \min_{\theta}$$

## Градиентный спуск

$$\omega[t + 1] = \omega[t] - \alpha[t] \nabla (L, w[t])$$

## Почему сложно обучать нейронную сеть

- Не всегда оптимизируемая функция дифференцируема

## Оптимизационная задача

$$L(x, \theta) = \sum_{(x_i, y_i) \in \text{train}} L(x_i, y_i, \theta) \rightarrow \min_{\theta}$$

## Градиентный спуск

$$\omega[t + 1] = \omega[t] - \alpha[t] \nabla (L, w[t])$$

## Почему сложно обучать нейронную сеть

- Не всегда оптимизируемая функция дифференцируема
- Даже если она дифференцируемая, то иногда производная бывает тривиальной (для кусочно-постоянных функций)

# Почему обучать нейронную сеть сложно

## Почему сложно обучать нейронную сеть

- Не всегда оптимизируемая функция дифференцируема

<sup>4</sup><https://www.willamette.edu/~gorr/classes/cs449/momrate.html>

# Почему обучать нейронную сеть сложно

## Почему сложно обучать нейронную сеть

- Не всегда оптимизируемая функция дифференцируема
- Даже если она дифференцируемая, то иногда производная бывает тривиальной (для кусочно-постоянных функций)

<sup>4</sup><https://www.willamette.edu/~gorr/classes/cs449/momrate.html>

# Почему обучать нейронную сеть сложно

## Почему сложно обучать нейронную сеть

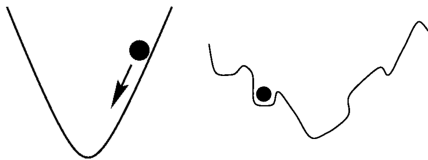
- Не всегда оптимизируемая функция дифференцируема
- Даже если она дифференцируемая, то иногда производная бывает тривиальной (для кусочно-постоянных функций)
- В большинстве случаев оптимизируемая функция не является выпуклой и имеет множество локальных минимумов

<sup>4</sup><https://www.willamette.edu/~gorr/classes/cs449/momrate.html>

# Почему обучать нейронную сеть сложно

## Почему сложно обучать нейронную сеть

- Не всегда оптимизируемая функция дифференцируема
- Даже если она дифференцируемая, то иногда производная бывает тривиальной (для кусочно-постоянных функций)
- В большинстве случаев оптимизируемая функция не является выпуклой и имеет множество локальных минимумов
- Даже если мы попали в глобальный минимум никто не обещает, что это хорошо, так как это может быть переобучением



4

<sup>4</sup><https://www.willamette.edu/~gorr/classes/cs449/momrate.html>

## Методы оптимизации

- Градиентный спуск

---

<sup>5</sup>Н. Robbins, and S. Monro. A Stochastic Approximation Method. 1951.



## Методы оптимизации

- Градиентный спуск
- Стохастический градиентный спуск <sup>5</sup>

<sup>5</sup>Н. Robbins, and S. Monro. A Stochastic Approximation Method. 1951.

## Методы оптимизации

- Градиентный спуск
- Стохастический градиентный спуск <sup>5</sup>
- Стохастический градиентный спуск с использованием мини-батча

<sup>5</sup>H. Robbins, and S. Monro. A Stochastic Approximation Method. 1951.

# Достоинства и недостатки градиентного спуска

## Достоинства

- Он работает
- Серьёзных альтернатив градиентному спуску нет

<sup>6</sup>Иллюстрация работы алгоритма при большом и маленьком шагах обучения»

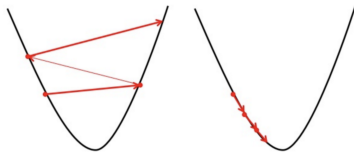
# Достоинства и недостатки градиентного спуска

## Достоинства

- Он работает
- Серьёзных альтернатив градиентному спуску нет

## Недостатки

- Выбор шага градиентного спуска довольно сложная задача
- Один и тот же шаг применяется ко всем параметрам при обновлении весов
- Нет гарантий сходимости в общем случае



6

<sup>6</sup>Иллюстрация работы алгоритма при большом и маленьком шагах обучения

## Momentum

$$v[t + 1] = \mu v[t] - \alpha[t] \nabla (L, w[t])$$

$$\omega[t + 1] = \omega[t] + v[t + 1]$$

<sup>7</sup><https://www.willamette.edu/~gorr/classes/cs449/momrate.html>

<sup>8</sup>Polyak, B.T. Some methods of speeding up the convergence of iteration methods. 1964.

## Momentum

$$v[t + 1] = \mu v[t] - \alpha[t] \nabla (L, w[t])$$

$$\omega[t + 1] = \omega[t] + v[t + 1]$$

- Типичное значение  $\mu = 0.9$

<sup>7</sup><https://www.willamette.edu/~gorr/classes/cs449/momrate.html>

<sup>8</sup>Polyak, B.T. Some methods of speeding up the convergence of iteration methods. 1964.

## Momentum

$$v[t + 1] = \mu v[t] - \alpha[t] \nabla (L, w[t])$$

$$\omega[t + 1] = \omega[t] + v[t + 1]$$

- Типичное значение  $\mu = 0.9$
- Направление шага алгоритма зависит от всех предыдущих значений градиентов взятых с экспоненциально убывающими коэффициентами

<sup>7</sup><https://www.willamette.edu/~gorr/classes/cs449/momrate.html>

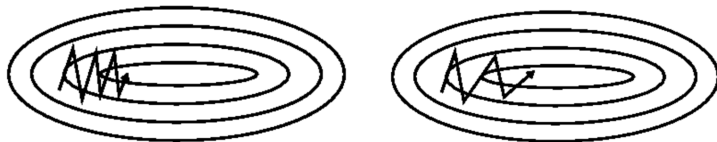
<sup>8</sup>Polyak, B.T. Some methods of speeding up the convergence of iteration methods. 1964.

## Momentum

$$v[t+1] = \mu v[t] - \alpha[t] \nabla (L, w[t])$$

$$\omega[t+1] = \omega[t] + v[t+1]$$

- Типичное значение  $\mu = 0.9$
- Направление шага алгоритма зависит от всех предыдущих значений градиентов взятых с экспоненциально убывающими коэффициентами



7

<sup>7</sup><https://www.willamette.edu/~gorr/classes/cs449/momrate.html>

<sup>8</sup>Polyak, B.T. Some methods of speeding up the convergence of iteration methods. 1964.



## Nesterov accelerated gradient

$$v[t+1] = \mu v[t] - \alpha[t] \nabla (L, w[t] + \mu v[t])$$

$$w[t+1] = w[t] + v[t+1]$$

<sup>9</sup>Nesterov, Y. A method of solving a convex programming problem with convergence rate  $O(1/\sqrt{k})$ . 1983.

## Nesterov accelerated gradient

$$v[t + 1] = \mu v[t] - \alpha[t] \nabla (L, w[t] + \mu v[t])$$

$$\omega[t + 1] = \omega[t] + v[t + 1]$$

- Главная идея — вычисление градиента в более релевантной точке
- Перед вычислением градиента мы уже имеем хорошее приближение точки, где мы будем на следующем шаге.

<sup>9</sup>Nesterov, Y. A method of solving a convex programming problem with convergence rate  $O(1/\sqrt{k})$ . 1983.

## Adagrad

$$g[t+1] = g[t] + \nabla(L, w[t]) \odot \nabla(L, w[t])$$

$$w[t+1] = w[t] - \frac{\alpha[t]}{\sqrt{g[t] + \varepsilon}} \odot \nabla(L, w[t])$$

<sup>10</sup>J. Duchi, E. Hazan, Y. Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. 2011

## Adagrad

$$g[t+1] = g[t] + \nabla(L, w[t]) \odot \nabla(L, w[t])$$

$$w[t+1] = w[t] - \frac{\alpha[t]}{\sqrt{g[t] + \varepsilon}} \odot \nabla(L, w[t])$$

- Главная идея — адаптирование шага градиента для различных направлений

<sup>10</sup>J. Duchi, E. Hazan, Y. Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. 2011

## Adagrad

$$g[t+1] = g[t] + \nabla(L, w[t]) \odot \nabla(L, w[t])$$

$$\omega[t+1] = \omega[t] - \frac{\alpha[t]}{\sqrt{g[t] + \varepsilon}} \odot \nabla(L, w[t])$$

- Главная идея — адаптирование шага градиента для различных направлений
- Веса, которые обновляются часто, имеют меньший шаг

<sup>10</sup>J. Duchi, E. Hazan, Y. Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. 2011

## Adagrad

$$g[t+1] = g[t] + \nabla(L, w[t]) \odot \nabla(L, w[t])$$

$$w[t+1] = w[t] - \frac{\alpha[t]}{\sqrt{g[t] + \varepsilon}} \odot \nabla(L, w[t])$$

- Главная идея — адаптирование шага градиента для различных направлений
- Веса, которые обновляются часто, имеют меньший шаг
- Главный недостаток, что  $g[t]$  возрастает

<sup>10</sup>J. Duchi, E. Hazan, Y. Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. 2011



## RMSprop

$$g[t+1] = \rho g[t] + (1 - \rho) \nabla(L, w[t]) \odot \nabla(L, w[t])$$
$$\omega[t+1] = \omega[t] - \frac{\alpha[t]}{\sqrt{g[t] + \varepsilon}} \odot \nabla(L, w[t])$$

<sup>11</sup>T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSEIRA: Neural Networks for Machine Learning. 2012.

## RMSprop

$$g[t+1] = \rho g[t] + (1 - \rho) \nabla (L, w[t]) \odot \nabla (L, w[t])$$

$$w[t+1] = w[t] - \frac{\alpha[t]}{\sqrt{g[t] + \varepsilon}} \odot \nabla (L, w[t])$$

- Такая же основная идея, но немного другая реализация

<sup>11</sup>T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSEIRA: Neural Networks for Machine Learning. 2012.



## RMSprop

$$g[t+1] = \rho g[t] + (1 - \rho) \nabla(L, w[t]) \odot \nabla(L, w[t])$$
$$w[t+1] = w[t] - \frac{\alpha[t]}{\sqrt{g[t] + \varepsilon}} \odot \nabla(L, w[t])$$

- Такая же основная идея, но немного другая реализация
- Типичное значение  $\rho = 0.9$

<sup>11</sup>T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSEIRA: Neural Networks for Machine Learning. 2012.

## Adadelta

$$g[t+1] = \rho g[t] + (1 - \rho) \nabla (L, w[t]) \odot \nabla (L, w[t])$$

$$\omega[t+1] = \omega[t] - \frac{\sqrt{d[t] + \varepsilon}}{\sqrt{g[t] + \varepsilon}} \odot \nabla (L, w[t])$$

$$d[t+1] = \rho d[t] + (1 - \rho)(\omega[t+1] - \omega[t]) \odot (\omega[t+1] - \omega[t])$$

<sup>12</sup>Zeiler, Matthew D. ADADELTA: An adaptive learning rate method. 2012.

## Adadelta

$$g[t+1] = \rho g[t] + (1 - \rho) \nabla (L, w[t]) \odot \nabla (L, w[t])$$

$$\omega[t+1] = \omega[t] - \frac{\sqrt{d[t] + \varepsilon}}{\sqrt{g[t] + \varepsilon}} \odot \nabla (L, w[t])$$

$$d[t+1] = \rho d[t] + (1 - \rho)(\omega[t+1] - \omega[t]) \odot (\omega[t+1] - \omega[t])$$

- Типичное значение  $\rho = 0.9$
- С единицами измерения теперь всё в порядке

<sup>12</sup>Zeiler, Matthew D. ADADELTA: An adaptive learning rate method. 2012.

**Algorithm 1:** *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation.  $g_t^2$  indicates the elementwise square  $g_t \odot g_t$ . Good default settings for the tested machine learning problems are  $\alpha = 0.001$ ,  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$  and  $\epsilon = 10^{-8}$ . All operations on vectors are element-wise. With  $\beta_1^t$  and  $\beta_2^t$  we denote  $\beta_1$  and  $\beta_2$  to the power  $t$ .

**Require:**  $\alpha$ : Stepsize

**Require:**  $\beta_1, \beta_2 \in [0, 1)$ : Exponential decay rates for the moment estimates

**Require:**  $f(\theta)$ : Stochastic objective function with parameters  $\theta$

**Require:**  $\theta_0$ : Initial parameter vector

$m_0 \leftarrow 0$  (Initialize 1<sup>st</sup> moment vector)

$v_0 \leftarrow 0$  (Initialize 2<sup>nd</sup> moment vector)

$t \leftarrow 0$  (Initialize timestep)

**while**  $\theta_t$  not converged **do**

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$  (Get gradients w.r.t. stochastic objective at timestep  $t$ )

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$  (Update biased first moment estimate)

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$  (Update biased second raw moment estimate)

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$  (Compute bias-corrected first moment estimate)

$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$  (Compute bias-corrected second raw moment estimate)

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$  (Update parameters)

**end while**

**return**  $\theta_t$  (Resulting parameters)

<sup>13</sup>Diederik, Kingma; Ba, Jimmy. Adam: A method for stochastic optimization. 2014.

---

**Algorithm 2:** *AdaMax*, a variant of Adam based on the infinity norm. See section 7.1 for details. Good default settings for the tested machine learning problems are  $\alpha = 0.002$ ,  $\beta_1 = 0.9$  and  $\beta_2 = 0.999$ . With  $\beta_1^t$  we denote  $\beta_1$  to the power  $t$ . Here,  $(\alpha/(1 - \beta_1^t))$  is the learning rate with the bias-correction term for the first moment. All operations on vectors are element-wise.

---

**Require:**  $\alpha$ : Stepsize

**Require:**  $\beta_1, \beta_2 \in [0, 1)$ : Exponential decay rates

**Require:**  $f(\theta)$ : Stochastic objective function with parameters  $\theta$

**Require:**  $\theta_0$ : Initial parameter vector

$m_0 \leftarrow 0$  (Initialize 1<sup>st</sup> moment vector)

$u_0 \leftarrow 0$  (Initialize the exponentially weighted infinity norm)

$t \leftarrow 0$  (Initialize timestep)

**while**  $\theta_t$  not converged **do**

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$  (Get gradients w.r.t. stochastic objective at timestep  $t$ )

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$  (Update biased first moment estimate)

$u_t \leftarrow \max(\beta_2 \cdot u_{t-1}, |g_t|)$  (Update the exponentially weighted infinity norm)

$\theta_t \leftarrow \theta_{t-1} - (\alpha/(1 - \beta_1^t)) \cdot m_t/u_t$  (Update parameters)

**end while**

**return**  $\theta_t$  (Resulting parameters)



14

---

<sup>14</sup><http://ruder.io/optimizing-gradient-descent>



# Прочие подходы и эвристики

- Nadam <sup>15</sup>
- Cyclical Learning Rates for Training Neural Networks <sup>16</sup>
- Radam <sup>17</sup>
- Lookahead optimizer <sup>18</sup>
- ...
- Тут может быть ссылка на вашу работу

---

<sup>15</sup> Timothy Dozat. Incorporating Nesterov Momentum into Adam. 2015.

<sup>16</sup> L. Smith. Cyclical Learning Rates for Training Neural Networks. 2019.

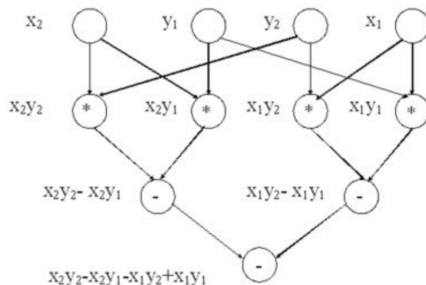
<sup>17</sup> L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, J. Han. On the Variance of the Adaptive Learning Rate and Beyond. 2019.

<sup>18</sup> M. Zhang, J. Lucas, G. Hinton, J. Ba. Lookahead Optimizer: k steps forward, 1 step back. 2019.



# Граф вычислений

- Нейронную сеть удобно представлять в виде ориентированного графа (очень часто ациклического, но не всегда)
- В листьях такого дерева находятся входные данные
- В остальных вершинах находятся операции





# Производная сложной функции

$$z_1 = z_1(y_1, y_2)$$

$$z_2 = z_2(y_1, y_2)$$

$$y_1 = y_1(x_1, x_2)$$

$$y_2 = y_2(x_1, x_2)$$

## Chain rule (производная сложной функции)

$$\frac{dz}{dx} = \begin{pmatrix} \frac{\partial z_1}{\partial x_1} & \frac{\partial z_1}{\partial x_2} \\ \frac{\partial z_2}{\partial x_1} & \frac{\partial z_2}{\partial x_2} \end{pmatrix} = \begin{pmatrix} \frac{\partial z_1}{\partial y_1} \frac{\partial y_1}{\partial x_1} + \frac{\partial z_1}{\partial y_2} \frac{\partial y_2}{\partial x_1} & \frac{\partial z_1}{\partial y_1} \frac{\partial y_1}{\partial x_2} + \frac{\partial z_1}{\partial y_2} \frac{\partial y_2}{\partial x_2} \\ \frac{\partial z_2}{\partial y_1} \frac{\partial y_1}{\partial x_1} + \frac{\partial z_2}{\partial y_2} \frac{\partial y_2}{\partial x_1} & \frac{\partial z_2}{\partial y_1} \frac{\partial y_1}{\partial x_2} + \frac{\partial z_2}{\partial y_2} \frac{\partial y_2}{\partial x_2} \end{pmatrix} = \begin{pmatrix} \frac{\partial z_1}{\partial y_1} & \frac{\partial z_1}{\partial y_2} \\ \frac{\partial z_2}{\partial y_1} & \frac{\partial z_2}{\partial y_2} \end{pmatrix} \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{pmatrix}$$
$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$$

## Постановка задачи

Дано:

$$x_0 \xrightarrow{W_1} x_1 \xrightarrow{W_2} x_2 \rightarrow \dots \xrightarrow{W_s} x_s, \text{ то есть } x_{i+1} = x_{i+1}(x_i, W_{i+1})$$

<sup>19</sup>D. Rumelhart, G. Hinton, R. Williams. Learning representations by back-propagating errors. 1986.

# Метод обратного распространения ошибки<sup>19</sup>

## Постановка задачи

Дано:

$$x_0 \xrightarrow{W_1} x_1 \xrightarrow{W_2} x_2 \rightarrow \dots \xrightarrow{W_s} x_s, \text{ то есть } x_{i+1} = x_{i+1}(x_i, W_{i+1})$$

Найти:

$$\frac{dx_s}{dW_1}, \frac{dx_s}{dW_2}, \dots, \frac{dx_s}{dW_s},$$

## Схема вычисления

0.  $\frac{dx_s}{dW_s}, \frac{dx_s}{dx_{s-1}}$

<sup>19</sup>D. Rumelhart, G. Hinton, R. Williams. Learning representations by back-propagating errors. 1986.

# Метод обратного распространения ошибки<sup>19</sup>

## Постановка задачи

Дано:

$$x_0 \xrightarrow{W_1} x_1 \xrightarrow{W_2} x_2 \rightarrow \dots \xrightarrow{W_s} x_s, \text{ то есть } x_{i+1} = x_{i+1}(x_i, W_{i+1})$$

Найти:

$$\frac{dx_s}{dW_1}, \frac{dx_s}{dW_2}, \dots, \frac{dx_s}{dW_s},$$

## Схема вычисления

$$0. \quad \frac{dx_s}{dW_s}, \frac{dx_s}{dx_{s-1}}$$

$$1. \quad \frac{dx_s}{dW_{s-1}} = \frac{dx_s}{dx_{s-1}} \frac{dx_{s-1}}{dW_{s-1}}, \quad \frac{dx_s}{dx_{s-2}} = \frac{dx_s}{dx_{s-1}} \frac{dx_{s-1}}{dx_{s-2}}$$

<sup>19</sup>D. Rumelhart, G. Hinton, R. Williams. Learning representations by back-propagating errors. 1986.

# Метод обратного распространения ошибки<sup>19</sup>

## Постановка задачи

Дано:

$$x_0 \xrightarrow{W_1} x_1 \xrightarrow{W_2} x_2 \rightarrow \dots \xrightarrow{W_s} x_s, \text{ то есть } x_{i+1} = x_{i+1}(x_i, W_{i+1})$$

Найти:

$$\frac{dx_s}{dW_1}, \frac{dx_s}{dW_2}, \dots, \frac{dx_s}{dW_s},$$

## Схема вычисления

$$0. \quad \frac{dx_s}{dW_s}, \frac{dx_s}{dx_{s-1}}$$

$$1. \quad \frac{dx_s}{dW_{s-1}} = \frac{dx_s}{dx_{s-1}} \frac{dx_{s-1}}{dW_{s-1}}, \quad \frac{dx_s}{dx_{s-2}} = \frac{dx_s}{dx_{s-1}} \frac{dx_{s-1}}{dx_{s-2}}$$

$$2. \quad \frac{dx_s}{dW_{s-2}} = \frac{dx_s}{dx_{s-2}} \frac{dx_{s-2}}{dW_{s-2}}, \quad \frac{dx_s}{dx_{s-3}} = \frac{dx_s}{dx_{s-2}} \frac{dx_{s-2}}{dx_{s-3}}$$

<sup>19</sup>D. Rumelhart, G. Hinton, R. Williams. Learning representations by back-propagating errors. 1986.

# Метод обратного распространения ошибки<sup>19</sup>

## Постановка задачи

Дано:

$$x_0 \xrightarrow{W_1} x_1 \xrightarrow{W_2} x_2 \rightarrow \dots \xrightarrow{W_s} x_s, \text{ то есть } x_{i+1} = x_{i+1}(x_i, W_{i+1})$$

Найти:

$$\frac{dx_s}{dW_1}, \frac{dx_s}{dW_2}, \dots, \frac{dx_s}{dW_s},$$

## Схема вычисления

0.  $\frac{dx_s}{dW_s}, \frac{dx_s}{dx_{s-1}}$

1.  $\frac{dx_s}{dW_{s-1}} = \frac{dx_s}{dx_{s-1}} \frac{dx_{s-1}}{dW_{s-1}}, \frac{dx_s}{dx_{s-2}} = \frac{dx_s}{dx_{s-1}} \frac{dx_{s-1}}{dx_{s-2}}$

2.  $\frac{dx_s}{dW_{s-2}} = \frac{dx_s}{dx_{s-2}} \frac{dx_{s-2}}{dW_{s-2}}, \frac{dx_s}{dx_{s-3}} = \frac{dx_s}{dx_{s-2}} \frac{dx_{s-2}}{dx_{s-3}}$

...

<sup>19</sup>D. Rumelhart, G. Hinton, R. Williams. Learning representations by back-propagating errors. 1986.

# Метод обратного распространения ошибки

## Схема вычисления

$$0. \quad \frac{dx_s}{dW_s}, \frac{dx_s}{dx_{s-1}}$$

$$1. \quad \frac{dx_s}{dW_{s-1}} = \frac{dx_s}{dx_{s-1}} \frac{dx_{s-1}}{dW_{s-1}}, \quad \frac{dx_s}{dx_{s-2}} = \frac{dx_s}{dx_{s-1}} \frac{dx_{s-1}}{dx_{s-2}}$$

$$2. \quad \frac{dx_s}{dW_{s-2}} = \frac{dx_s}{dx_{s-2}} \frac{dx_{s-2}}{dW_{s-2}}, \quad \frac{dx_s}{dx_{s-3}} = \frac{dx_s}{dx_{s-2}} \frac{dx_{s-2}}{dx_{s-3}}$$

...

$$i. \quad \frac{dx_s}{dW_{s-i}} = \frac{dx_s}{dx_{s-i}} \frac{dx_{s-i}}{dW_{s-i}}, \quad \frac{dx_s}{dx_{s-i-1}} = \frac{dx_s}{dx_{s-i}} \frac{dx_{s-i}}{dx_{s-i-1}}$$



# Метод обратного распространения ошибки

## Схема вычисления

$$0. \quad \frac{dx_s}{dW_s}, \frac{dx_s}{dx_{s-1}}$$

$$1. \quad \frac{dx_s}{dW_{s-1}} = \frac{dx_s}{dx_{s-1}} \frac{dx_{s-1}}{dW_{s-1}}, \quad \frac{dx_s}{dx_{s-2}} = \frac{dx_s}{dx_{s-1}} \frac{dx_{s-1}}{dx_{s-2}}$$

$$2. \quad \frac{dx_s}{dW_{s-2}} = \frac{dx_s}{dx_{s-2}} \frac{dx_{s-2}}{dW_{s-2}}, \quad \frac{dx_s}{dx_{s-3}} = \frac{dx_s}{dx_{s-2}} \frac{dx_{s-2}}{dx_{s-3}}$$

...

$$i. \quad \frac{dx_s}{dW_{s-i}} = \frac{dx_s}{dx_{s-i}} \frac{dx_{s-i}}{dW_{s-i}}, \quad \frac{dx_s}{dx_{s-i-1}} = \frac{dx_s}{dx_{s-i}} \frac{dx_{s-i}}{dx_{s-i-1}}$$

...

$$s-1. \quad \frac{dx_s}{dW_1} = \frac{dx_s}{dx_1} \frac{dx_1}{dW_1}$$

## Заключение

Главное правило: при обратном распространении градиента на каждом шаге необходимо считать производную по весам и по входу!



# Метод обратного распространения ошибки

- Метод обратного распространения полностью соответствует парадигме объектно-ориентированного программирования



# Метод обратного распространения ошибки

- Метод обратного распространения полностью соответствует парадигме объектно-ориентированного программирования
- На каждом шаге необходимо вычислять градиент по весам и по входу



# Метод обратного распространения ошибки

- Метод обратного распространения полностью соответствует парадигме объектно-ориентированного программирования
- На каждом шаге необходимо вычислять градиент по весам и по входу
- Сложность алгоритма линейна относительно глубины цепочки зависимостей



# Метод обратного распространения ошибки

- Метод обратного распространения полностью соответствует парадигме объектно-ориентированного программирования
- На каждом шаге необходимо вычислять градиент по весам и по входу
- Сложность алгоритма линейна относительно глубины цепочки зависимостей
- Метод позволяет вычислять производные довольно сложных функций, в графе которых нет циклов



# Метод обратного распространения ошибки

- Метод обратного распространения полностью соответствует парадигме объектно-ориентированного программирования
- На каждом шаге необходимо вычислять градиент по весам и по входу
- Сложность алгоритма линейна относительно глубины цепочки зависимостей
- Метод позволяет вычислять производные довольно сложных функций, в графе которых нет циклов
- В случае, если в графе есть циклы, то существуют модификации алгоритма для разных специальных случаев



- Важно инициализировать веса не нулями



- Важно инициализировать веса не нулями
- Для ReLU используйте инициализацию He, а для симметричных функций инициализацию Xavier



- Важно инициализировать веса не нулями
- Для ReLU используйте инициализацию He, а для симметричных функций инициализацию Xavier
- От параметров и выбора оптимизатора зависит скорость и качество обучения





- Важно инициализировать веса не нулями
- Для ReLU используйте инициализацию He, а для симметричных функций инициализацию Xavier
- От параметров и выбора оптимизатора зависит скорость и качество обучения
- Исследования по методам оптимизации для нейронных сетей активно продолжаются



- Важно инициализировать веса не нулями
- Для ReLU используйте инициализацию He, а для симметричных функций инициализацию Xavier
- От параметров и выбора оптимизатора зависит скорость и качество обучения
- Исследования по методам оптимизации для нейронных сетей активно продолжаются
- На практике часто бывает, что SGD+momentum работает более стабильно



- Важно инициализировать веса не нулями
- Для ReLU используйте инициализацию He, а для симметричных функций инициализацию Xavier
- От параметров и выбора оптимизатора зависит скорость и качество обучения
- Исследования по методам оптимизации для нейронных сетей активно продолжаются
- На практике часто бывает, что SGD+momentum работает более стабильно
- Метод обратного распространения ошибки — эффективная схема вычисления градиента



Спасибо за внимание!

