Введение в искусственный интеллект. Современное компьютерное зрение

Тема: Обучение нейронных сетей

Бабин Д.Н., Иванов И.Е., Петюшко А.А.

кафедра Математической Теории Интеллектуальных Систем

План лекции

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

Этапы подготовки модели к обучению

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

Методы инициализации весов нейронной сети

- 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

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

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

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

Пример многослойной сети

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

Взрывающийся градиент (exploding gradient)

Если $W_i = diag(1.5, \ldots, 1.5)$, то $y_i = (1.5)^i x$ и

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

Затухающий градиент (vanishing gradient)

Если $W_i = diag(0.5, \ldots, 0.5)$, то $y_i = (0.5)^i x$ и

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

Параметры распределений

Вопросы

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

Параметры распределений

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

Подсчет дисперсии

Лемма

Для независимых случайных величин X и Y выполнено

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

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

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

Подсчет дисперсии

Лемма

Для независимых случайных величин X и Y выполнено

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

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

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

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 подсчета дисперсии

Пример

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

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

Пример подсчета дисперсии

Пример

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

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

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

Более продвинутые методы инициализации весов нейронной сети

- Lecun-инициализация [1] (n_i — количество входов (нейронов) i-го слоя)

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

[1] LeCun Y. A. et al. Efficient backprop. 1998.

[2] **Xavier** Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. 2010.

Более продвинутые методы инициализации весов нейронной сети

- Lecun-инициализация [1] (n_i — количество входов (нейронов) i-го слоя)

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

- Xavier-инициализация [2]

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

[1] LeCun Y. A. et al. Efficient backprop. 1998.
[2] **Xavier** Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. 2010.

Подсчет дисперсии для ReLU сетей

Продолжим выкладки:

$$D(y_i) = n_{in}(E[x_j]^2 D(w_i) + E[w_{ij}]^2 D(x_j) + D(w_{ij})D(x_j)) =$$

$$= n_{in}(E[x_j]^2 D(w_{ij}) + 0 \cdot D(x_j) + D(w_{ij})D(x_j)) =$$

$$= n_{in}(E[x_j]^2 D(w_{ij}) + D(w_{ij})D(x_j)) = n_{in} D(w_{ij}) E[x_j^2]$$

Подсчет дисперсии для ReLU сетей

Продолжим выкладки:

$$D(y_i) = n_{in}(E[x_j]^2 D(w_i) + E[w_{ij}]^2 D(x_j) + D(w_{ij})D(x_j)) =$$

$$= n_{in}(E[x_j]^2 D(w_{ij}) + 0 \cdot D(x_j) + D(w_{ij})D(x_j)) =$$

$$= n_{in}(E[x_j]^2 D(w_{ij}) + D(w_{ij})D(x_j)) = n_{in}D(w_{ij})E[x_j^2]$$

Подсчитаем как меняется дисперсия при использовании функции активации ReLU

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

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

Более продвинутые методы инициализации весов нейронной сети

- Инициализация для ReLU сетей [3]

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

[3] 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 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

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

Методы оптимизации нейронной сети

Оптимизационная задача

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

Градиентный спуск

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

Методы оптимизации нейронной сети

Оптимизационная задача

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

Градиентный спуск

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

Почему сложно обучать нейронную сеть

- Не всегда оптимизируемая функция дифференцируема

Методы оптимизации нейронной сети

Оптимизационная задача

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

Градиентный спуск

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

Почему сложно обучать нейронную сеть

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

Почему сложно обучать нейронную сеть

- Не всегда оптимизируемая функция дифференцируема

[4] https://www.willamette.edu/~gorr/classes/cs449/momrate.html

Почему сложно обучать нейронную сеть

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

[4] https://www.willamette.edu/~gorr/classes/cs449/momrate.html

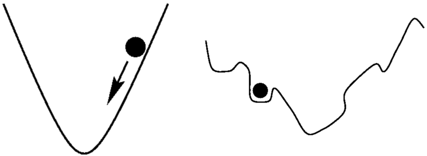
Почему сложно обучать нейронную сеть

- 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

[4] https://www.willamette.edu/~gorr/classes/cs449/momrate.html

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]

[4] https://www.willamette.edu/~gorr/classes/cs449/momrate.html

Методы оптимизации

- Градиентный спуск

[5] H. Robbins, and S. Monro. A Stochastic Approximation Method. 1951.

Методы оптимизации

- Градиентный спуск
- Стохастический градиентный спуск [5]

[5] H. Robbins, and S. Monro. A Stochastic Approximation Method. 1951.

Методы оптимизации

- Градиентный спуск
- Стохастический градиентный спуск [5]
- Стохастический градиентный спуск с использованием мини-батча

[5] H. Robbins, and S. Monro. A Stochastic Approximation Method. 1951.

Достоинства и недостатки градиентного спуска

Достоинства

- Он работает
- Серьёзных альтернатив градиентному спуску нет

[6] Иллюстрация работы алгоритма при большом и маленьком шагах обучения

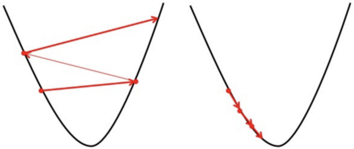
Достоинства и недостатки градиентного спуска

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]

[6] Иллюстрация работы алгоритма при большом и маленьком шагах обучения

Моментум [8]

Momentum

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

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

[7] https://www.willamette.edu/~gorr/classes/cs449/momrate.html

[8] Polyak, B.T. Some methods of speeding up the convergence of iteration methods. 1964.

Моментум [8]

Momentum

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

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

- Типичное значение $\mu = 0.9$

[7] https://www.willamette.edu/~gorr/classes/cs449/momrate.html

[8] Polyak, B.T. Some methods of speeding up the convergence of iteration methods. 1964.

Моментум [8]

Momentum

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

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

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

[7] https://www.willamette.edu/~gorr/classes/cs449/momrate.html

[8] Polyak, B.T. Some methods of speeding up the convergence of iteration methods. 1964.

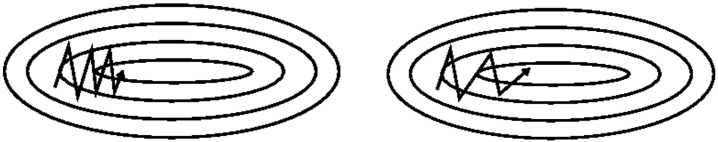
Моментум [8]

Momentum

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

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

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



[7]

[7] https://www.willamette.edu/~gorr/classes/cs449/momrate.html

[8] Polyak, B.T. Some methods of speeding up the convergence of iteration methods. 1964.

Градиент Нестерова [9]

Nesterov accelerated gradient

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

[9] Nesterov, Y. A method of solving a convex programming problem with convergence rate O(1/sqr(k)). 1983.

Градиент Нестерова [9]

Nesterov accelerated gradient

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

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

[9] Nesterov, Y. A method of solving a convex programming problem with convergence rate O(1/sqr(k)). 1983.

Adaptive Gradient Algorithm [10]

Adagrad

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

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

[10] J. Duchi, E. Hazan, Y. Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. 2011

Adaptive Gradient Algorithm [10]

Adagrad

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

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

- Главная идея — адаптирование шага градиента для различных направлений

[10] J. Duchi, E. Hazan, Y. Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. 2011

Adaptive Gradient Algorithm [10]

Adagrad

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

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

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

[10] J. Duchi, E. Hazan, Y. Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. 2011

Adaptive Gradient Algorithm [10]

Adagrad

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

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

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

[10] J. Duchi, E. Hazan, Y. Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. 2011

Root Mean Square Propagation [11]

RMSprop

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

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

[11] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning. 2012.

Root Mean Square Propagation [11]

RMSprop

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

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

- Такая же основная идея, но немного другая реализация

[11] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning. 2012.

Root Mean Square Propagation [11]

RMSprop

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

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

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

[11] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning. 2012.

Adadelta [12]

Adadelta

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

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

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

[12] Zeiler, Matthew D. ADADELTA: An adaptive learning rate method. 2012.

Adadelta [12]

Adadelta

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

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

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

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

[12] Zeiler, Matthew D. ADADELTA: An adaptive learning rate method. 2012.

Adam [13]

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 β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_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)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

[13] Diederik, Kingma; Ba, Jimmy. Adam: A method for stochastic optimization. 2014.

Adamax

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 β_1^t we denote β_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: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $u_0 \leftarrow 0$ (Initialize the exponentially weighted infinity norm)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_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 θ_t (Resulting parameters)

Визуализация различных оптимизаторов

14

[14] http://ruder.io/optimizing-gradient-descent

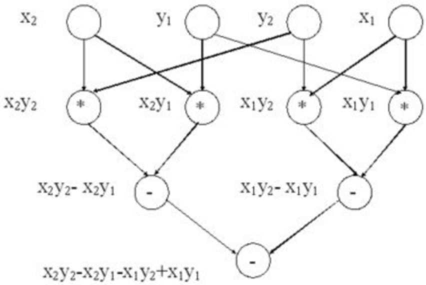
Прочие подходы и эвристики

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

[15] Timothy Dozat. Incorporating Nesterov Momentum into Adam. 2015.
[16] L. Smith. Cyclical Learning Rates for Training Neural Networks. 2019.
[17] 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.
[18] 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 ацикличного, 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)

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

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

Метод обратного распространения ошибки [19]

Постановка задачи

Дано:

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

[19] D. Rumelhart, G. Hinton, R. Williams. Learning representations by back-propagating errors. 1986.

Метод обратного распространения ошибки [19]

Постановка задачи

Дано:

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

Найти:

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

Схема вычисления

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

[19] D. Rumelhart, G. Hinton, R. Williams. Learning representations by back-propagating errors. 1986.

Метод обратного распространения ошибки [19]

Постановка задачи

Дано:

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

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

[19] D. Rumelhart, G. Hinton, R. Williams. Learning representations by back-propagating errors. 1986.

Метод обратного распространения ошибки [19]

Постановка задачи

Дано:

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

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. $\frac{dx_s}{dW_{s-1}} = \frac{dx_s}{dx_{s-1}} \frac{dx_{s-1}}{dW_{s-1}}, \quad \frac{dx_s}{dx_{s-2}} = \frac{dx_s}{dx_{s-1}} \frac{dx_{s-1}}{dx_{s-2}}$
2. $\frac{dx_s}{dW_{s-2}} = \frac{dx_s}{dx_{s-2}} \frac{dx_{s-2}}{dW_{s-2}}, \quad \frac{dx_s}{dx_{s-3}} = \frac{dx_s}{dx_{s-2}} \frac{dx_{s-2}}{dx_{s-3}}$

[19] D. Rumelhart, G. Hinton, R. Williams. Learning representations by back-propagating errors. 1986.

Метод обратного распространения ошибки [19]

Постановка задачи

Дано:

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

Найти:

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

Схема вычисления

0. $\frac{dx_s}{dW_s}, \frac{dx_s}{dx_{s-1}}$
1. $\frac{dx_s}{dW_{s-1}} = \frac{dx_s}{dx_{s-1}} \frac{dx_{s-1}}{dW_{s-1}}, \quad \frac{dx_s}{dx_{s-2}} = \frac{dx_s}{dx_{s-1}} \frac{dx_{s-1}}{dx_{s-2}}$
2. $\frac{dx_s}{dW_{s-2}} = \frac{dx_s}{dx_{s-2}} \frac{dx_{s-2}}{dW_{s-2}}, \quad \frac{dx_s}{dx_{s-3}} = \frac{dx_s}{dx_{s-2}} \frac{dx_{s-2}}{dx_{s-3}}$

...

[19] D. Rumelhart, G. Hinton, R. Williams. Learning representations by back-propagating errors. 1986.

Метод обратного распространения ошибки

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

...

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

Метод обратного распространения ошибки

Схема вычисления

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

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

...

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

...

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

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 соотвествует парадигме объектно-ориентированного программирования

Метод обратного распространения ошибки

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

Метод обратного распространения ошибки

- Метод обратного распространения полностью соотвествует 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 соотвествует 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 соотвествует 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

Заключение

- Важно инициализировать веса не нулями

Заключение

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

Заключение

- 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

Заключение

- 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

Заключение

- 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

Заключение

- 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!