Deep Learning. Введение в полносвязные нейросети. Алгоритмы оптимизации

Урок 2

Егор Конягин

Содержание

Логистическая регрессия. Повторение

Логистическая регрессия. Повторение

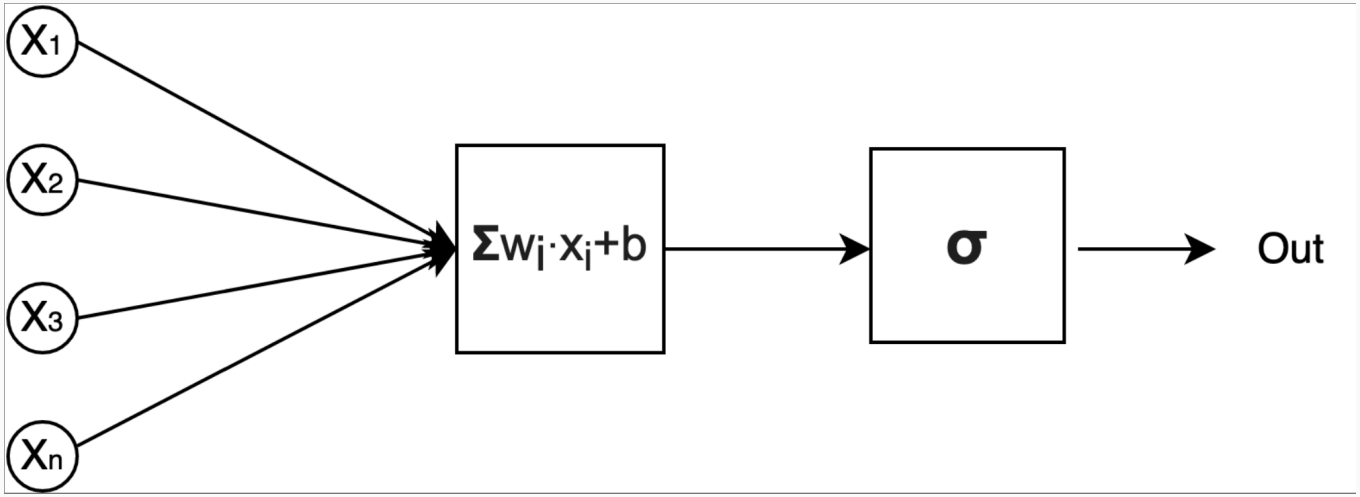


Рис. 1: Модель логистической регрессии

Логистическая регрессия. Повторение - II

Реализация логистической регрессии состоит из трёх шагов: инициализации весов, шага forward prop и шага backprop. Это можно описать следующими уравнениями:

$$z = w^T x + b \tag{1}$$

$$\hat{y} = a = \sigma(z). \tag{2}$$

$$J(\hat{y}, y) = -\sum_i y \log \hat{y} + (1 - y) \cdot \log(1 - \hat{y}) \tag{3}$$

Логистическая регрессия. Повторение - III

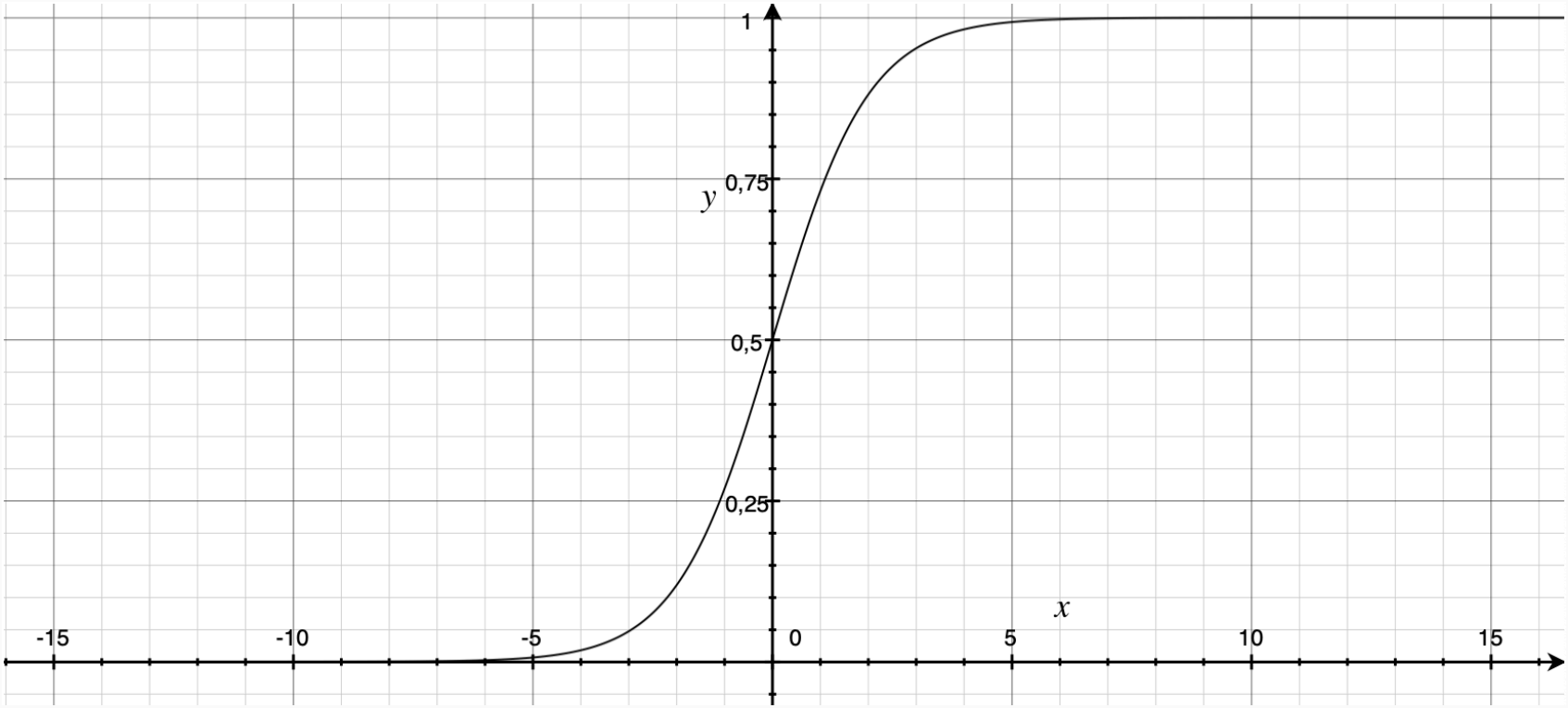


Рис. 2: График сигмоидной функции

Нейронные сети

Постановка задачи

Рассмотрим датасет для задачи классификации: два признака (x, y—координаты) и цвет (то есть предсказываемая величина) (синий или красный). Наша нейросеть будет предсказывать цвет (1 - синий, 0 - красный).

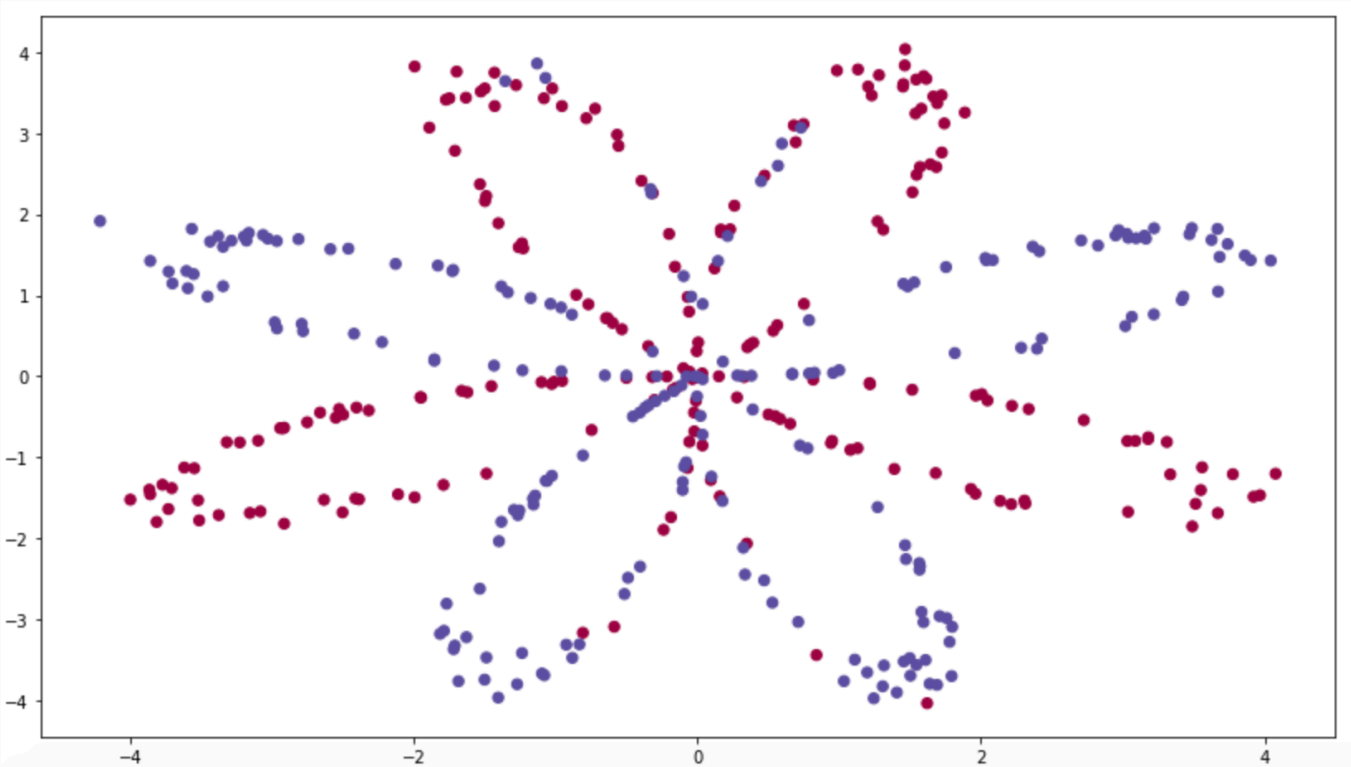


Рис. 3: Датасет точек

Постановка задачи - II

Мы уже обсуждали, что логистическая регрессия справляется только с линейно разделимыми данными. В данном случае ее лучший результат имеет следующий вид:

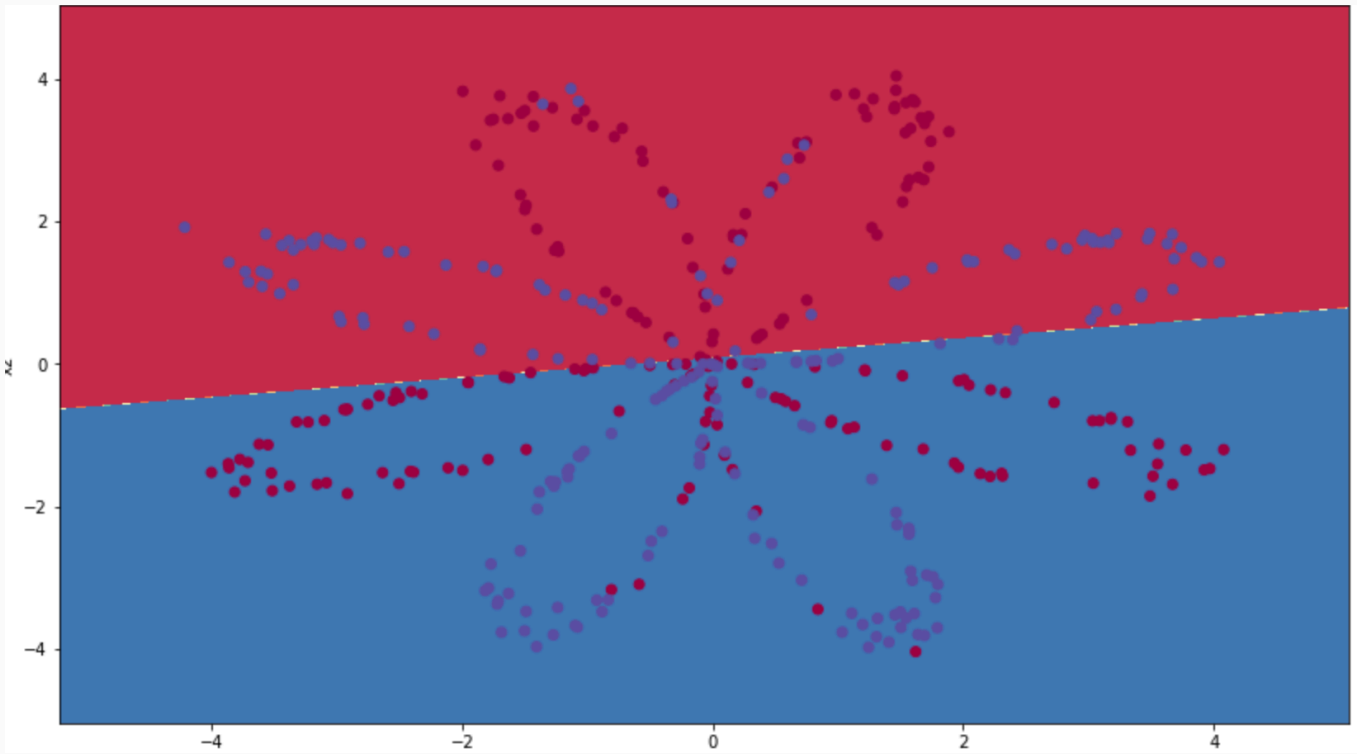


Рис. 4: Логистическая регрессия. Качество (accuracy): 47%

Описание архитектуры

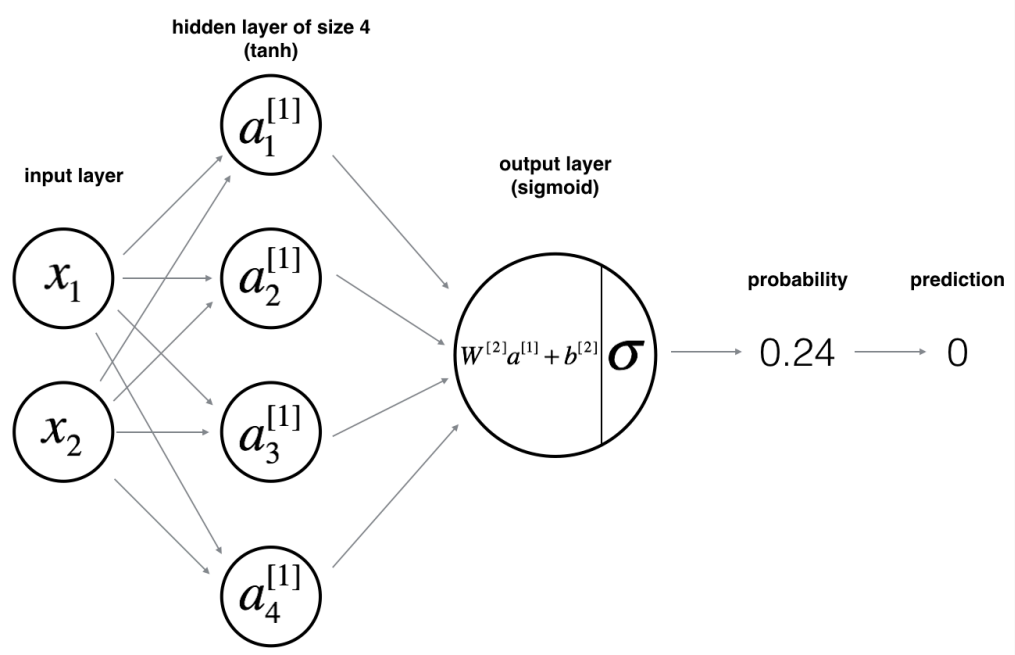


Рис. 5: Вариант архитектуры нейронной сети. Источник: Stanford University (Andrew Ng)

Пример с большим количеством слоев

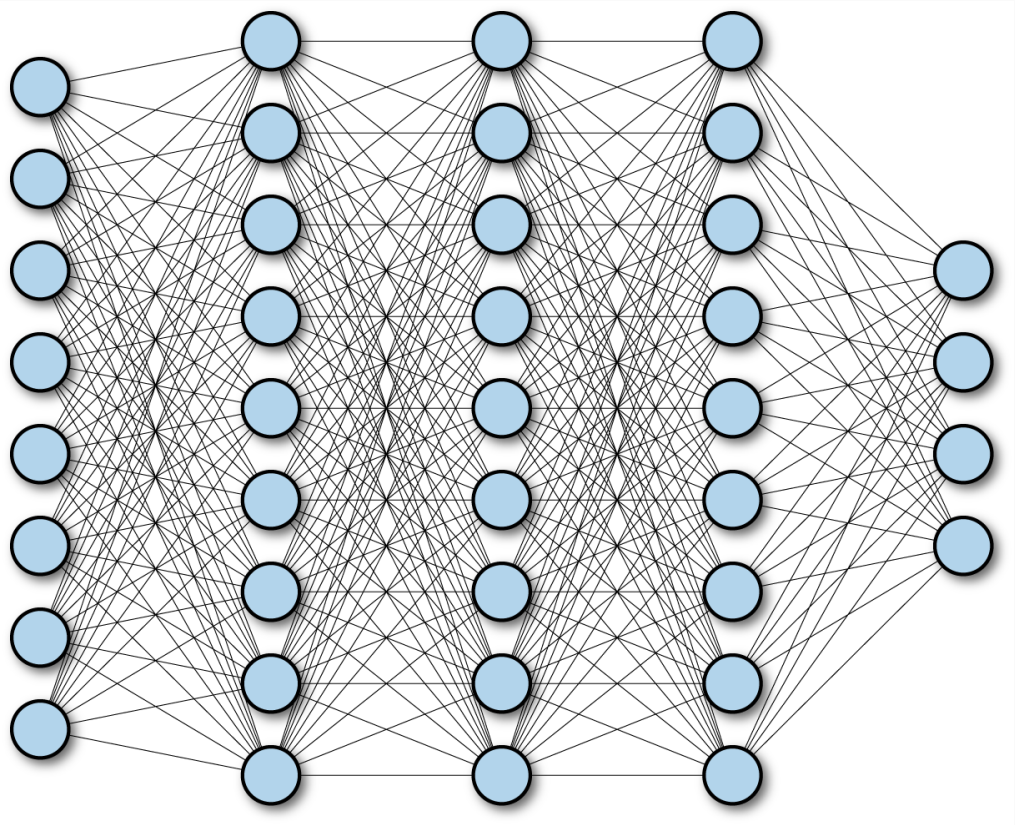


Рис. 6: Вариант архитектуры нейронной сети с большим количеством слоев. Источник: O'Reilly

Математическое описание нейросети

Математическое описание нейросети

Обучение нейронной сети состоит из тех же пунктов, что и у логистической регрессии:

1. Инициализация весов;
2. Начало цикла
    2.1 Шаг forward propagation;
    2.2 Шаг backward propagation;
    2.3 Обновление весов;
3. Завершение обучения.

Разница состоит в том, что forward propagation и back propagation описываются другими уравнениями (их вид определяется архитектурой модели).

Шаг forward propagation

Для первого слоя нейросети уравнения выглядят так:

$$z^{[1]} = X \cdot W^{[1]T} + b^{[1]} \tag{4}$$

$$a^{[1]} = \sigma^{[1]}(z^{[1]}). \tag{5}$$

Для любого слоя нейросети из L слоев (общий вид):

$$z^{[l]} = a^{[l-1]} \cdot W^{[l]T} + b^{[l]}, \quad l \in [1, L] \tag{6}$$

$$a^{[l]} = \sigma^{[l]}(z^{[l]}), \quad l \in [1, L] \tag{7}$$

$$\hat{y} = a^{[L]} \text{ - вывод нейросети.} \tag{8}$$

$$Loss = -\frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)}\log\left(\hat{y}^{(i)}\right) + (1-y^{(i)})\log\left(1-y^{(i)}\right)\right) \tag{9}$$

$\sigma^{[L]}$ - это нелинейные функции (**функции активации**).

Согласование размерностей

Каждый слой нейросети может произвольно изменять размерность входного вектора. Размер выхода l-ого слоя определяется кол-вом нейронов в этом слое: 1 нейрон генерирует ровно 1 число.

Если в слое 10 нейронов, то в выходе этого слоя будет вектор с 10 координатами. Пример:

$$dim X = (n_{objects}, n_{features}), \quad n_{objects} \quad \text{может быть любым!}$$

$$dim\ a^{[1]} = 10 \rightarrow X_{(n_{objects}, n_{features})} \cdot W_{(?,?)} = z^{[1]}_{(n_{objects}, 10)}$$

Согласование размерностей

Каждый слой нейросети может произвольно изменять размерность входного вектора. Размер выхода l-ого слоя определяется кол-вом нейронов в этом слое: 1 нейрон генерирует ровно 1 число.

Если в слое 10 нейронов, то в выходе этого слоя будет вектор с 10 координатами. Пример:

$$dim X = (n_{objects}, n_{features}), \quad n_{objects} \quad \text{может быть любым!}$$

$$dim\ a^{[1]} = 10 \rightarrow X_{(n_{objects}, n_{features})} \cdot W^T_{(?,?)} = z^{[1]}_{(n_{objects}, 10)}$$

Размеры матрицы W можно рассчитать из правила матричного произведения:

$$dim\ W^T = (n_{features}, 10) \rightarrow dim\ W = (10, n_{features})$$

Согласование размерностей. Вывод

В общем случае, **если обозначить количество нейронов в l-ом слое за N_l**, то размерности матриц параметров выглядят следующим образом:

$$dim\ W^{[l]} = (N_l, N_{l-1}), \tag{10}$$

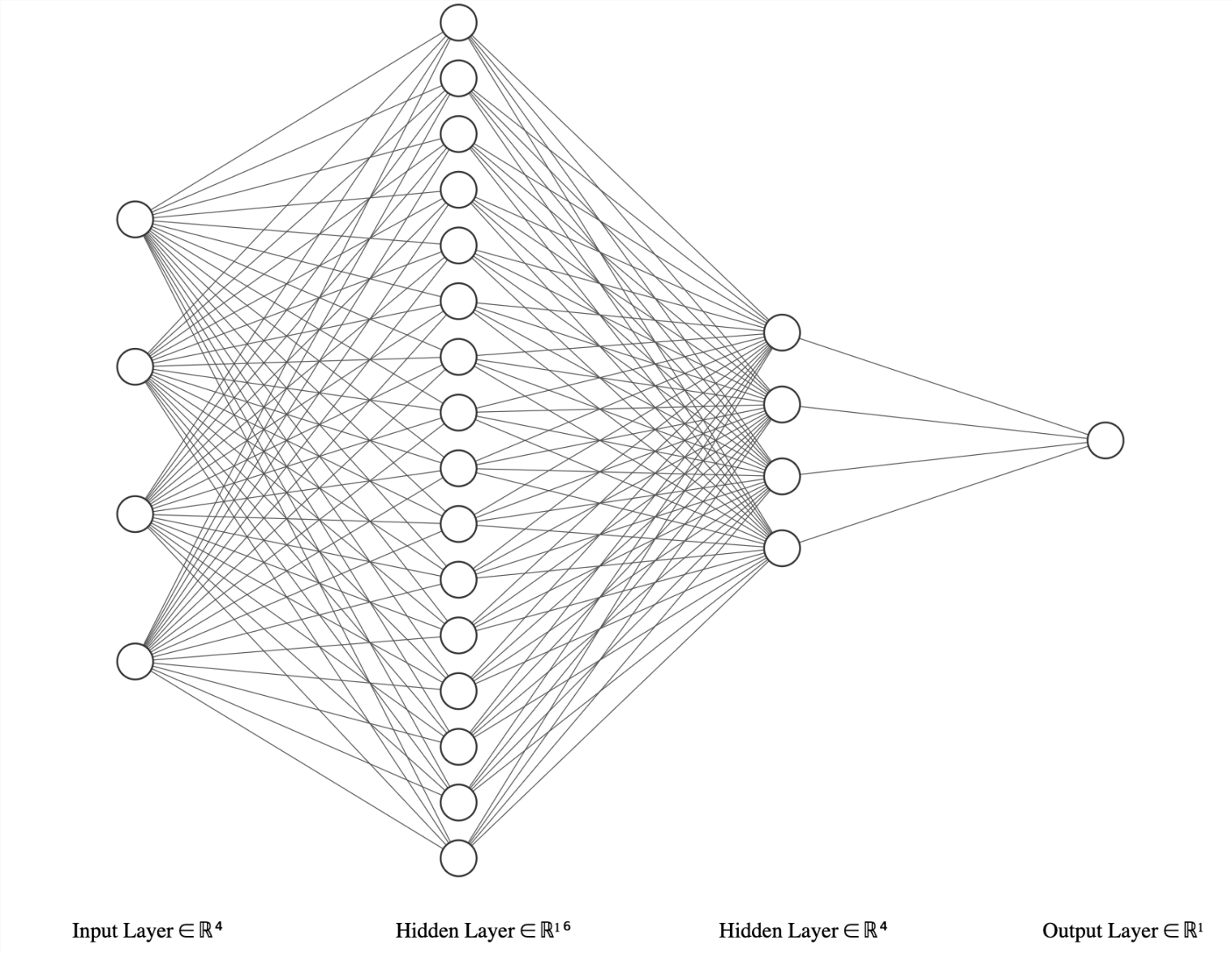
$$dim\ b^{[l]} = N_l. \tag{11}$$

Тогда легко можно рассчитать общее количество параметров нейросети:

$$N_{params} = \sum_{l=1}^{L} \left(size\ W^{[l]} + size\ b^{[l]} \right), \tag{12}$$

$$size\ W^{[l]} = N_l \times N_{l-1}, \quad size\ b^{[l]} = N_l.$$

Упражнение


Input Layer $\in \mathbb{R}^4$
Hidden Layer $\in \mathbb{R}^{16}$
Hidden Layer $\in \mathbb{R}^4$
Output Layer $\in \mathbb{R}^1$

Функции активации

Универсальная теорема об аппроксимации

Универсальная теорема об аппроксимации - теорема, звучащая следующим образом:

Пусть $\varphi : \mathcal{R} \to \mathcal{R}$ - это некая ограниченная, непрерывная и неконстантная функция. Пусть I_m - это m-мерный замкнутый гиперкуб вида $[0, 1]^m$.

Тогда для любого $\varepsilon > 0$ и для любой функции $f \in C(I_m)$ найдется целое число N и набор чисел $v_i, b_i \in \mathcal{R}$ и набор действительнозначных векторов w_i таких что:

$$F(x) = \sum_{i=1}^{N} v_i \varphi \left(w_i^T x + b_i \right), \tag{13}$$

$$|F(x) - f(x)| < \varepsilon. \tag{14}$$

Функции активации

Функции активации по сути делают возможным обучение нейросетей. Благодаря им нейросети способны аппроксимировать любую функцию!

Ниже представлены популярные функции активации:

- $\tanh(x) = \frac{e^x - 1}{e^x + 1}$;
- $ReLU(x) = \max(0, x)$;
- $sigmoid(x) = \frac{1}{1+e^{-x}}$

Функции активации обязательно нелинейные! В противном случае нейросеть ”схлопнется”в простую линейную модель (логистическую регрессию).

Алгоритмы оптимизации

Постановка задачи

Оптимизация - это процесс подбора параметров функции с целью ее минимизации. Рассмотрим формальную постановку задачи оптимизации:

$$L = L(y, \hat{y}(w, b)) \tag{15}$$

$$w^*, b^* = \arg\min L(y, \hat{y})|_{y=const} - \text{оптимальные параметры} \tag{16}$$

Проблема градиентного спуска

Рассмотрим функцию $z = x^2 - y^2$:

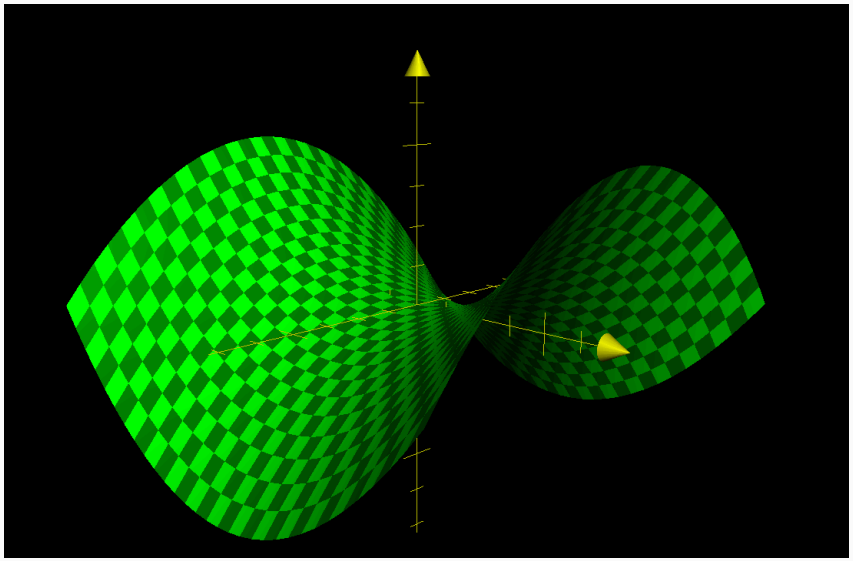


Рис. 7: График седловой функции вблизи точки (0,0)

Подсчитаем градиент данной функции:

$$\nabla z_{x,y} = (2x, -2y)^T \tag{17}$$

Видно, что в точке $(0, 0)$ $\nabla z_{(0,0)} = (0, 0)^T$. Таким образом, попав в эту точку, параметры перестанут обновляться!

Функция Розенброка

Функция Розенброка - это невыпуклая функция с одним глобальным минимумом. Она используется для оценки качества алгоритмов оптимизации. Формулой она задается следующим образом:

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2 \tag{18}$$

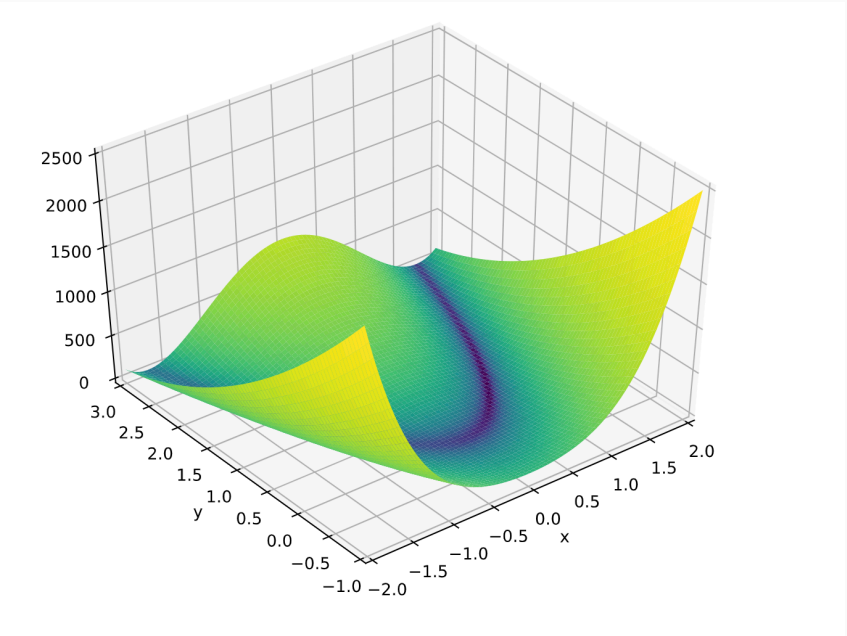


Рис. 8: График функции Розенброка

Методы оптимизации. Обзор

В задачах глубокого обучения используются градиентные методы оптимизации. Это значит, что функция потерь должна быть непрерывно дифференцируемой (на практике, их можно использовать и для кусочно-непрерывных функций).

Сегодня мы рассмотрим следующие (наиболее популярные в современном DL) методы:

- SGD - stochastic gradient descent (стохастический градиентный спуск);
- Mini batch GD - градиентный спуск по подвыборке;
- Momentum GD - градиентный спуск с импульсом;
- RMSProp (root mean square propagation);
- AdaM - Adaptive momentum GD.

Stochastic gradient descent. Mini-batch gradient descent

SGD. Описание

SGD - это градиентный спуск, при котором градиент считается не по всей выборке данных, а только используя один объект, случайно выбранный:

$$L = \sum_{i=1}^{N} \mathcal{L}(y^{(i)}, \hat{y}^{(i)}) \quad \rightarrow \quad \tilde{L} = \mathcal{L}(y^{(i^*)}, \hat{y}^{(i^*)}) \tag{19}$$

$$\frac{\partial L}{\partial w} \quad \rightarrow \quad \frac{\partial \tilde{L}}{\partial w} \tag{20}$$

$$\frac{\partial L}{\partial b} \quad \rightarrow \quad \frac{\partial \tilde{L}}{\partial b} \tag{21}$$

Mini-batch gradient descent

Mini-batch GD - это градиентный спуск, при котором градиент считается не по всей выборке данных, а по подвыборке объектов фиксированного размера (этот размер называется batch size, а сама подвыборка - batch):

$$L = \sum_{i=1}^{N} \mathcal{L}(y^{(i)}, \hat{y}^{(i)}) \quad \rightarrow \quad \tilde{L} = \sum_{j=1}^{batch\_size} \mathcal{L}(y^{(j)}, \hat{y}^{(j)}) \tag{22}$$

$$\frac{\partial J}{\partial w} \quad \rightarrow \quad \frac{\partial \tilde{J}}{\partial w} \tag{23}$$

$$\frac{\partial L}{\partial b} \quad \rightarrow \quad \frac{\partial \tilde{L}}{\partial b} \tag{24}$$

Mini-batch gradient descent vs SGD

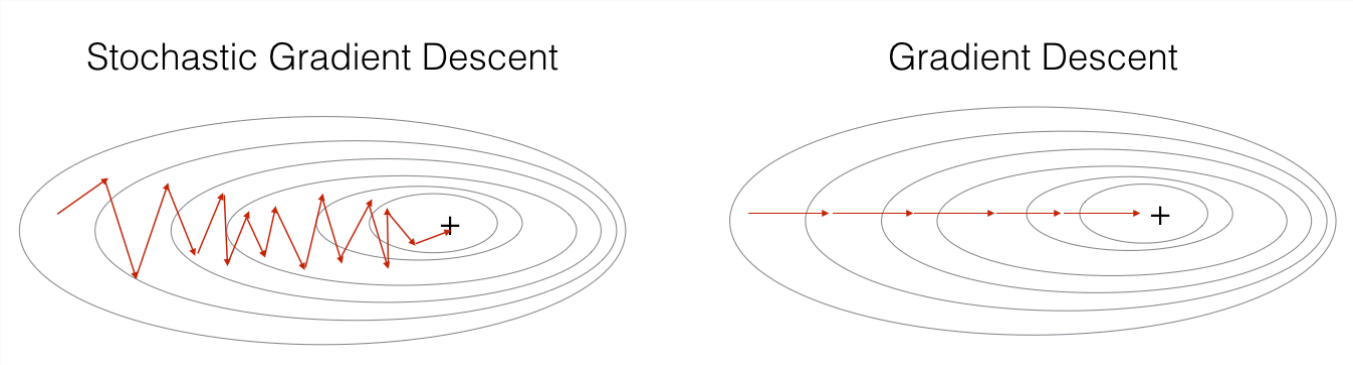


Рис. 9: SGD Vs. GD. Источник: Stanford University

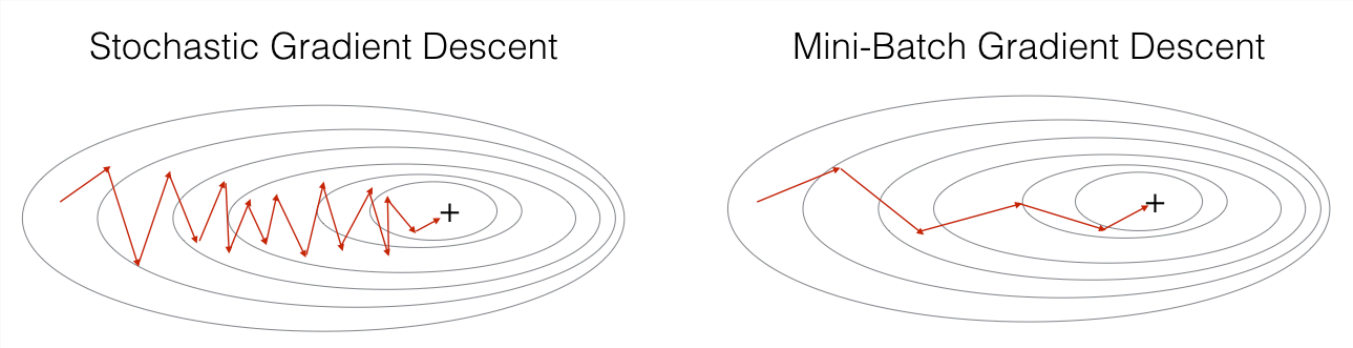


Рис. 10: Mini-batch GD Vs. GD. Источник: Stanford University

Понятие скользящего среднего

Скользящее среднее (exponentially weighted average) - это вариант аппроксимации среднего арифметического некой величины. Пусть есть последовательность $\theta = (\theta_1, \theta_2, \cdots \theta_n)$ неких измеренных величин. Тогда скользящее среднее рассчитывается по следующим формулам:

$$v_0 = 0 \tag{25}$$

$$v_1 = 0.9 \cdot v_0 + 0.1 \cdot \theta_1 \tag{26}$$

$$v_2 = 0.9 \cdot v_1 + 0.1 \cdot \theta_2 \tag{27}$$

$$v_t = \beta \cdot v_{t-1} + (1 - \beta) \cdot \theta_t \tag{28}$$

Коррекция смещения: проблема смещения возникает при малых значениях t для v_t: пусть $v_0 = 0$, тогда

$$v_1 = 0 + 0.1 \cdot \theta_1 = 0.1 \cdot \theta_1 << \theta_1 \tag{29}$$

Решение:

$$v_t \quad \rightarrow \quad \frac{v_t}{1 - \beta^t} \tag{30}$$

Momentum GD: RMSProp, AdaM

Momentum GD

Идея алгоритма: считать скользящие средние для градиентов, и использовать уже эти векторы, а не сами градиенты для обновления параметров. Суть его работы на картинке ниже:

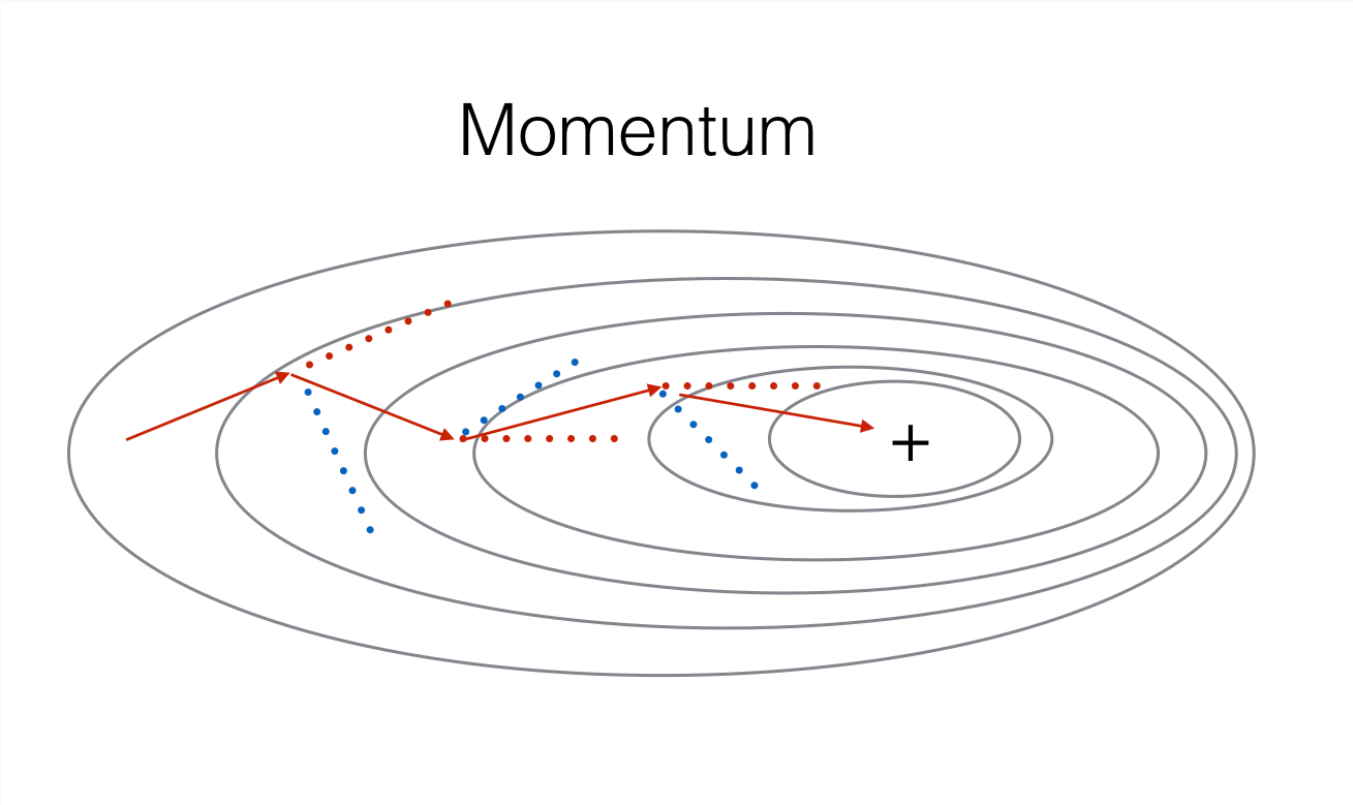


Рис. 11: Оптимизация функции с помощью MGD. Источник: Stanford University

Momentum GD. Математическое описание

Теперь рассмотрим уравнения, описывающие этот тип оптимизации. Реализация: считаем градиент на одном образце (SGD)...

$$dw = ... \quad db = ...$$

Далее считаем скользящее среднее:

$$V_{dw} = \beta \cdot V_{dw} + (1 - \beta) \cdot dw \tag{31}$$

$$V_{db} = \beta \cdot V_{db} + (1 - \beta) \cdot db \tag{32}$$

Далее происходит обновление весов:

$$w = w - \alpha \cdot V_{dw} \tag{33}$$

$$b = b - \alpha \cdot V_{db} \tag{34}$$

NB: β - это тоже гиперпараметр (характерные значения: 0.9)

RMSProp

Алгоритм RMSProp эффективно оптимизирует функции, в которых дисперсия значений по одной оси сильно отличается от дисперсии по другой.

Рассмотрим его реализацию:

- Инициализация: $S_{dw} = 0 \quad S_{db=0}$
- t-ый шаг

$$S_{dw} = \beta \cdot S_{dw} + (1 - \beta) \cdot (dw)^2 \tag{35}$$

$$S_{db} = \beta \cdot S_{db} + (1 - \beta) \cdot (db)^2 \tag{36}$$

- Update:

$$w = w - \frac{\alpha}{\sqrt{S_{dw}}} \cdot \frac{\partial \mathcal{L}}{\partial w} \tag{37}$$

AdaM

AdaM - это алгоритм, комбинирующий и RMSProp, и Momentum GD. Рассмотрим его реализацию:

- Инициализация значений:

$$V_{dw} = 0, \quad S_{dw} = 0, \quad V_{db} = 0, \quad S_{db} = 0; \tag{38}$$

- t-ый шаг: $dw, db = ...,$

$$V_{dw} = \beta_1 \cdot V_{dw} + (1 - \beta_1) \cdot dw, \quad S_{dw} = \beta_2 \cdot S_{dw} + (1 - \beta_2) \cdot (dw)^2 \tag{39}$$

$$V_{db} = \beta_1 \cdot V_{db} + (1 - \beta_1) \cdot db, \quad S_{db} = \beta_2 \cdot S_{db} + (1 - \beta_2) \cdot (db)^2 \tag{40}$$

- теперь применим коррекцию смещения:

$$V_{dw}^{correct} = \frac{V_{dw}}{1 - \beta_1^t}, \quad S_{dw} = \frac{S_{dw}}{1 - \beta_2^t}$$

$$V_{db}^{correct} = \frac{V_{db}}{1 - \beta_1^t}, \quad S_{db} = \frac{S_{db}}{1 - \beta_2^t}$$

AdaM - II

Далее производим обновление параметров:

$$w = w - \alpha \frac{V_{dw}^{correct}}{\sqrt{S_{dw}^{correct} + \varepsilon}} \tag{41}$$

$$b = b - \alpha \frac{V_{db}^{correct}}{\sqrt{S_{db}^{correct} + \varepsilon}} \tag{42}$$

Как видно из описания реализации, этот алгоритм содержит много гиперпараметров: $\alpha, \beta_1, \beta_2, \varepsilon$. Типичные значения для β_1 — это 0.9, для β_2 — это 0.999.