Искусственные нейронные сети для обработки текстов

Трифонов Владислав

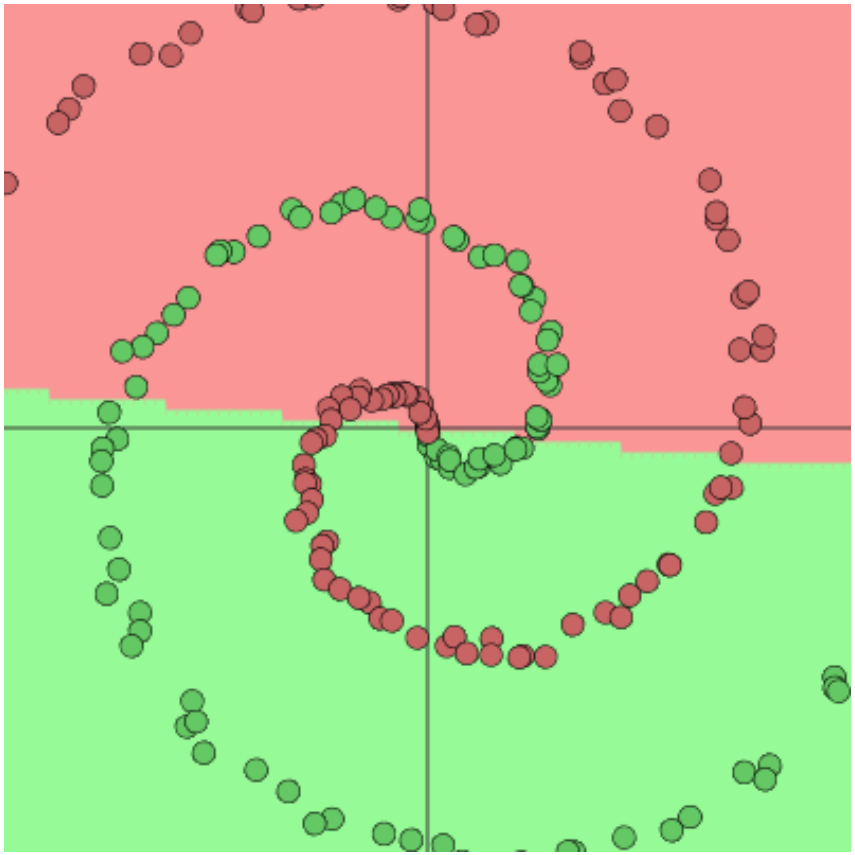
22 сентября 2020 г.

Введение

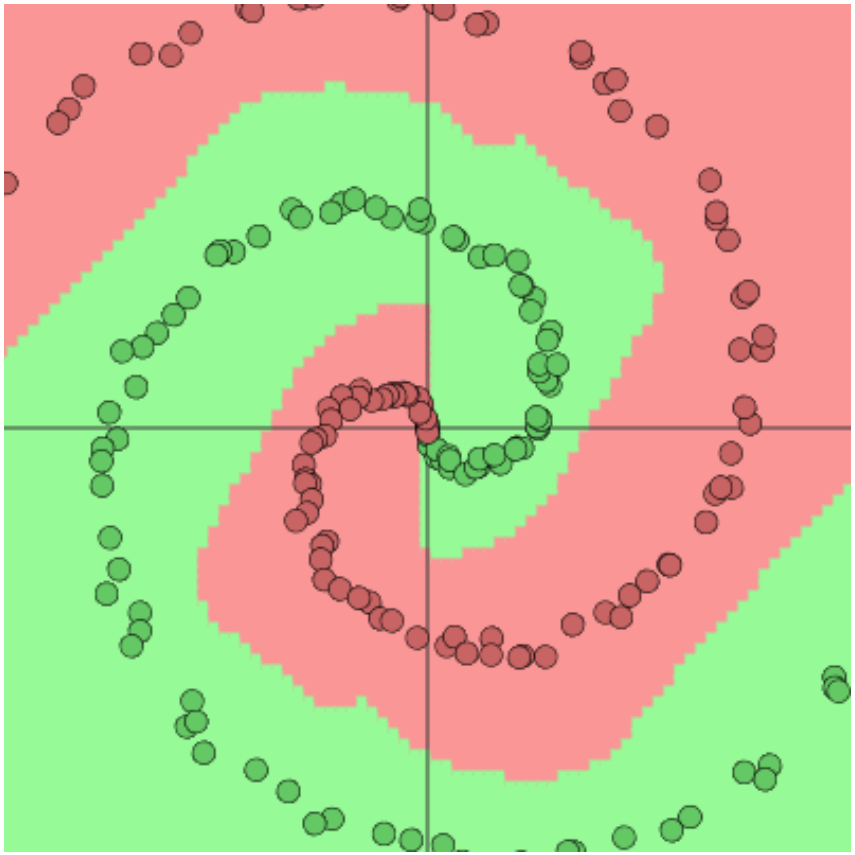
Нейронные сети – мощные алгоритмы машинного обучения, широко используемые при решении задач автоматической обработки текстов, изображений, видео и звуков.

В отличие от традиционных методов ML, они позволяют моделировать сложные зависимости в данных, автоматически строя признаковое пространство.

Введение



Линейный классификатор

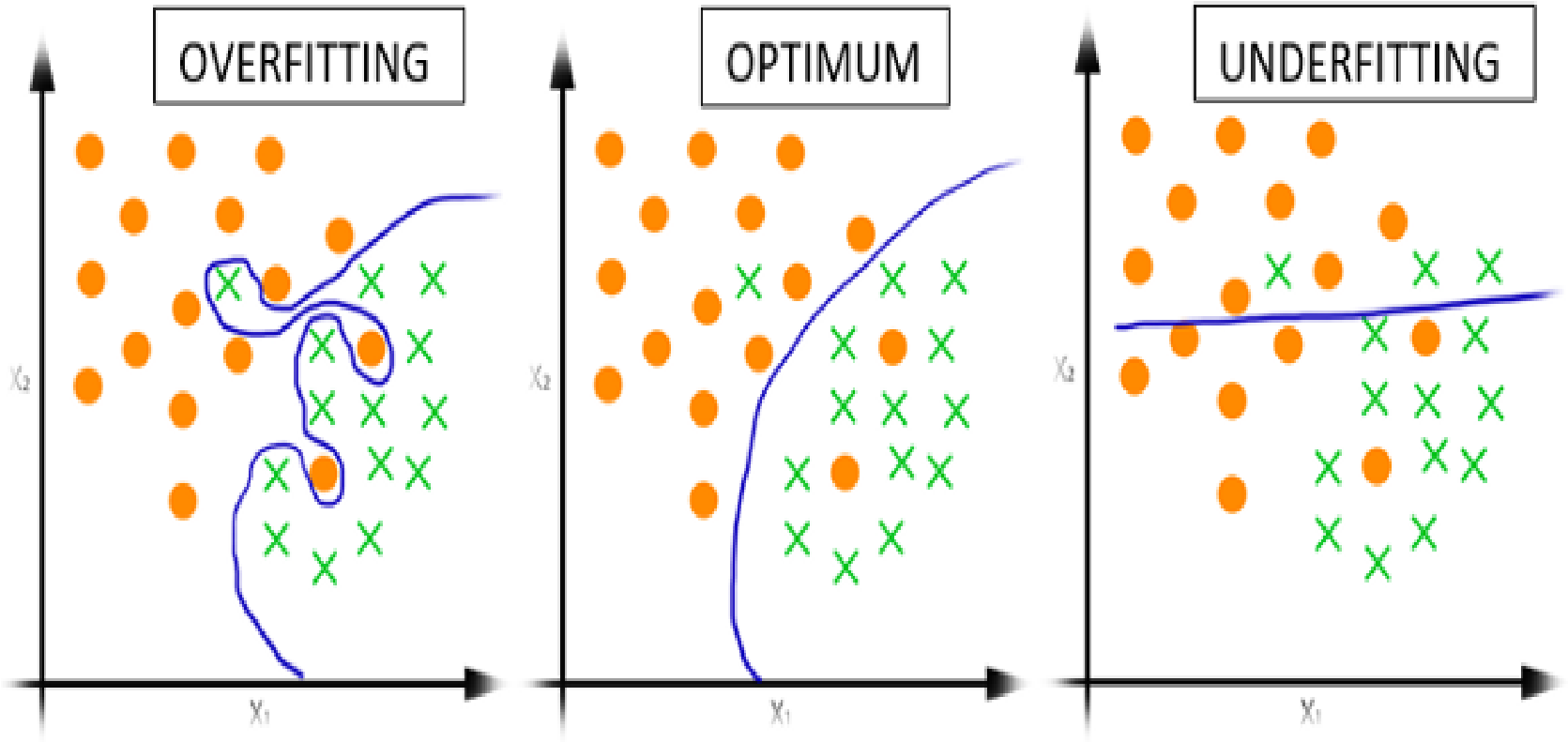


Нейронная сеть с одним скрытым слоем (tanh)

[0] cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

Введение

Чем “выразительнее” модель, тем больше шансов переобучиться!



[0] https://medium.com/@srjoglekar246/overfitting-and-human-behavior-5186df1e7d19

Обучение с учителем

Пусть

X – множество объектов

Y – множество ответов

$\tilde{y} : X \to Y$ – неизвестная зависимость

Дано

$\{x_1, ..., x_n\} \subset X$ – обучающая выборка

$\tilde{y}_i = \tilde{y}(x_i) \in Y$ – ответы на обучающей выборке

Найти

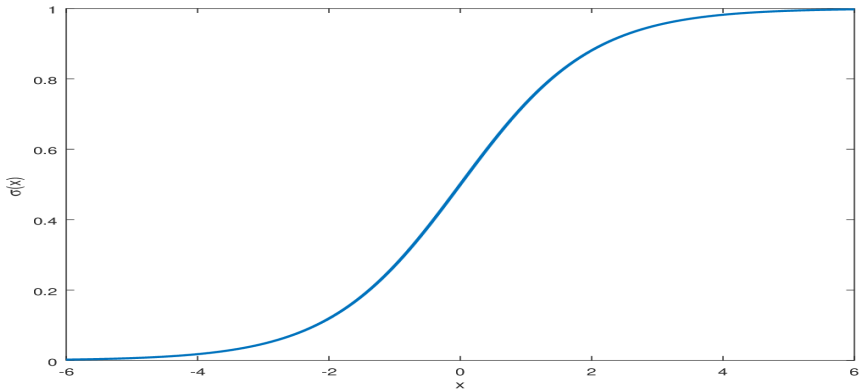
$y : X \to Y$ – алгоритм, решающую функцию, приближающую $\tilde{y}$ на всем множестве X

Логистическая регрессия

Дано

$\{\mathbf{x_1}, ..., \mathbf{x_n}\} \subset \mathbb{R}^d$ – обучающая выборка

$\tilde{y}_i = \tilde{y}(x_i) \in \{0, 1\}$ – ответы на обучающей выборке



Логистическая функция (сигмоида)

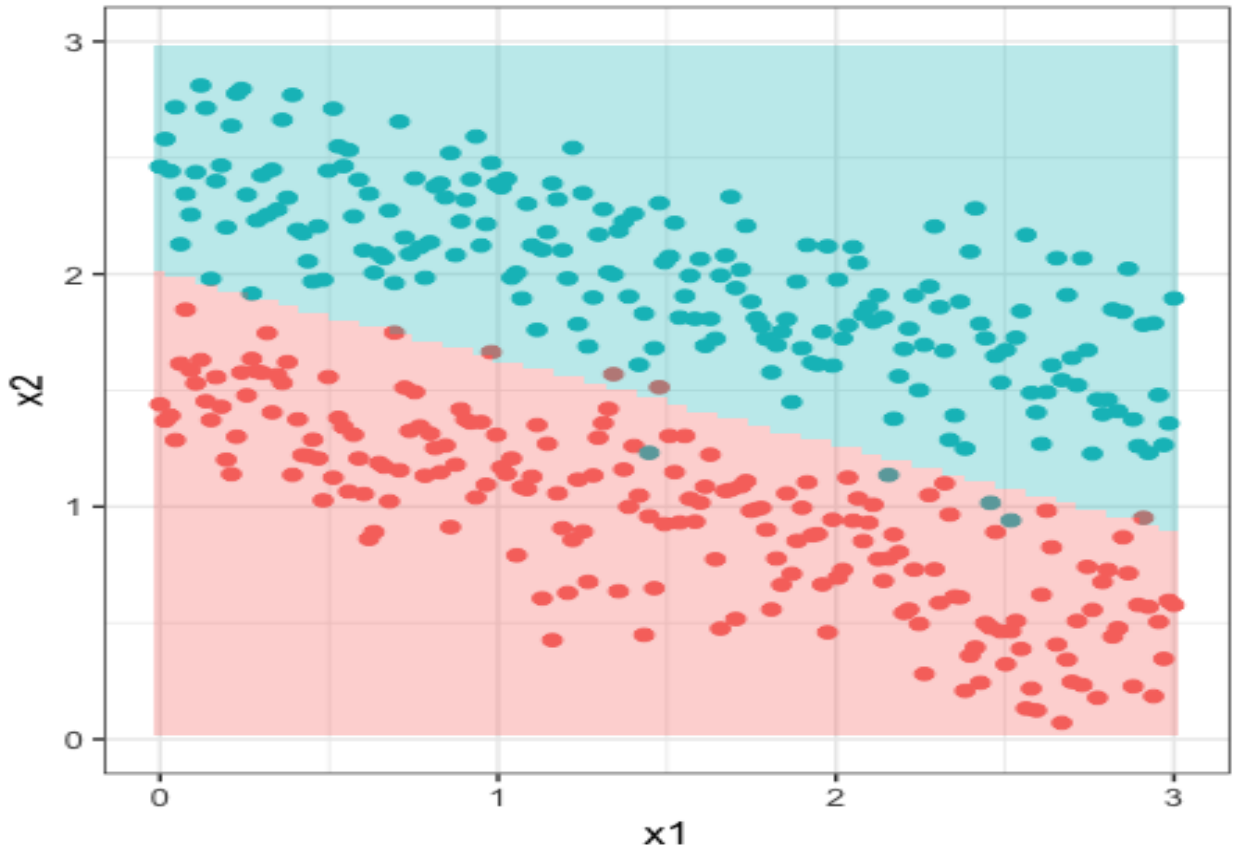
Решение

$$y = \sigma(\mathbf{w}^T \cdot \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T \cdot \mathbf{x})} \in (0, 1)$$

$$\mathbf{w} = \operatorname{argmin}_{\mathbf{w}} L(y, \tilde{y})$$

$$L(y, \tilde{y}) = -\frac{1}{n} \sum_{i=1}^{n} \tilde{y}_i \log y(\mathbf{x_i}) + (1 - \tilde{y}_i) \log(1 - y(\mathbf{x_i}))$$

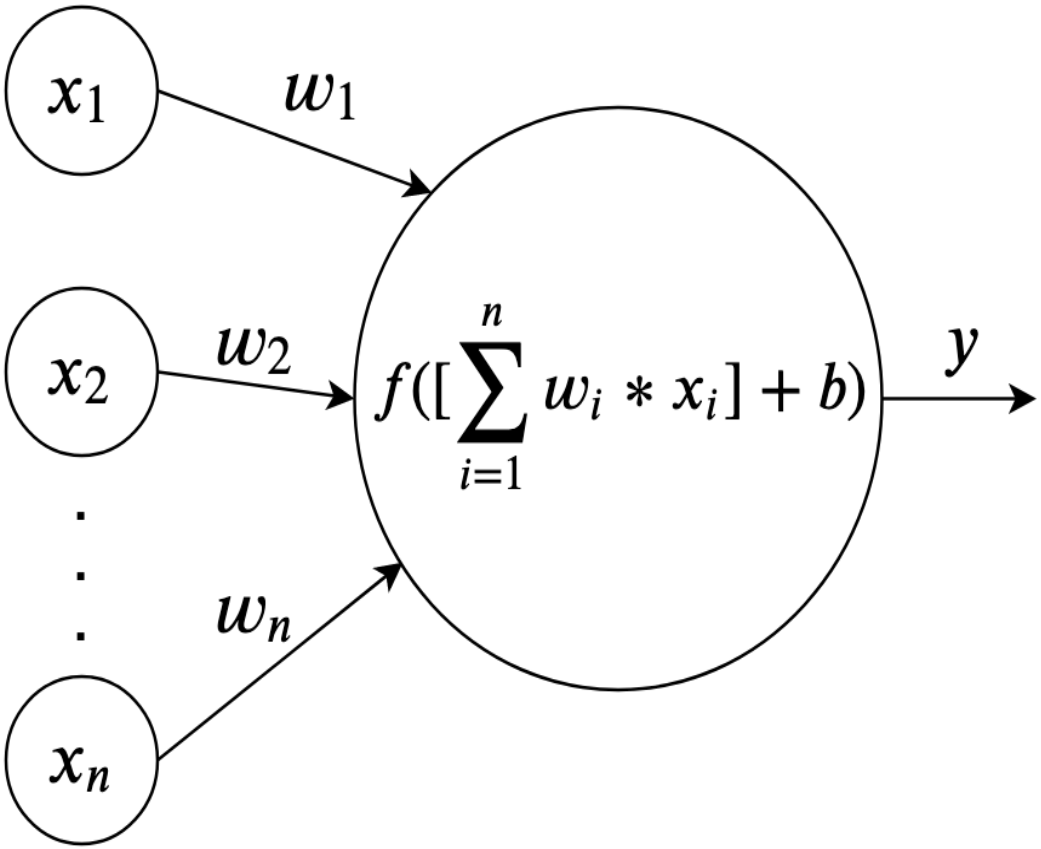
Логистическая регрессия



Решение $y(x) = 0.5$ – разделяющая гиперплоскость

[0] https://towardsdatascience.com/logistic-regression-from-scratch-in-r-b5b122fd8e83

Модель нейрона

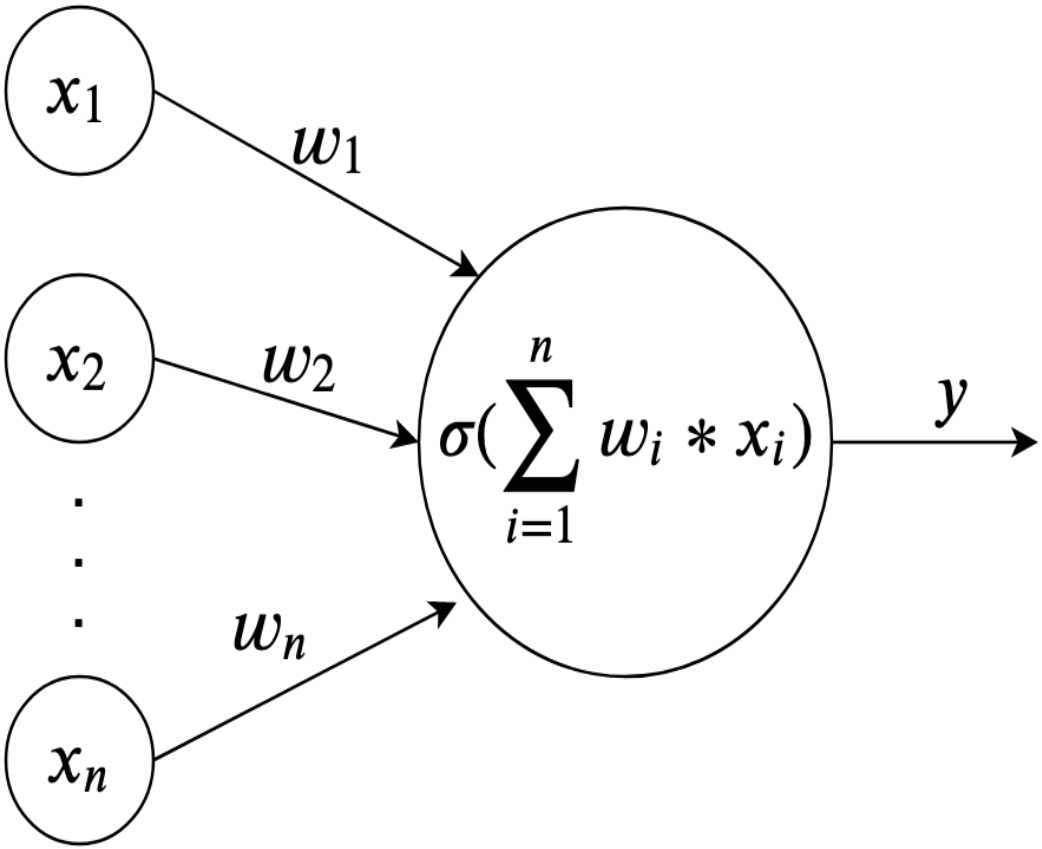


- W. McCulloch, W. Pitts – 1943
- $y = f(\mathbf{w}^T \cdot \mathbf{x} + b)$
- $\mathbf{w} \in \mathbb{R}^n$ – обучаемые веса связей
- $b \in \mathbb{R}$ – смещение (bias)
- f – функция активации (обычно нелинейная)

Модель нейрона

- Нейрон линейно комбинирует входные признаки и с помощью функции активации генерирует число – сигнал, который можно принять за ответ модели или использовать его в качестве нового признака
- Линейные модели (линейная регрессия, линейный классификатор, логистическая регрессия) – простейшие нейронные сети с одним нейроном

Логистическая регрессия как нейрон

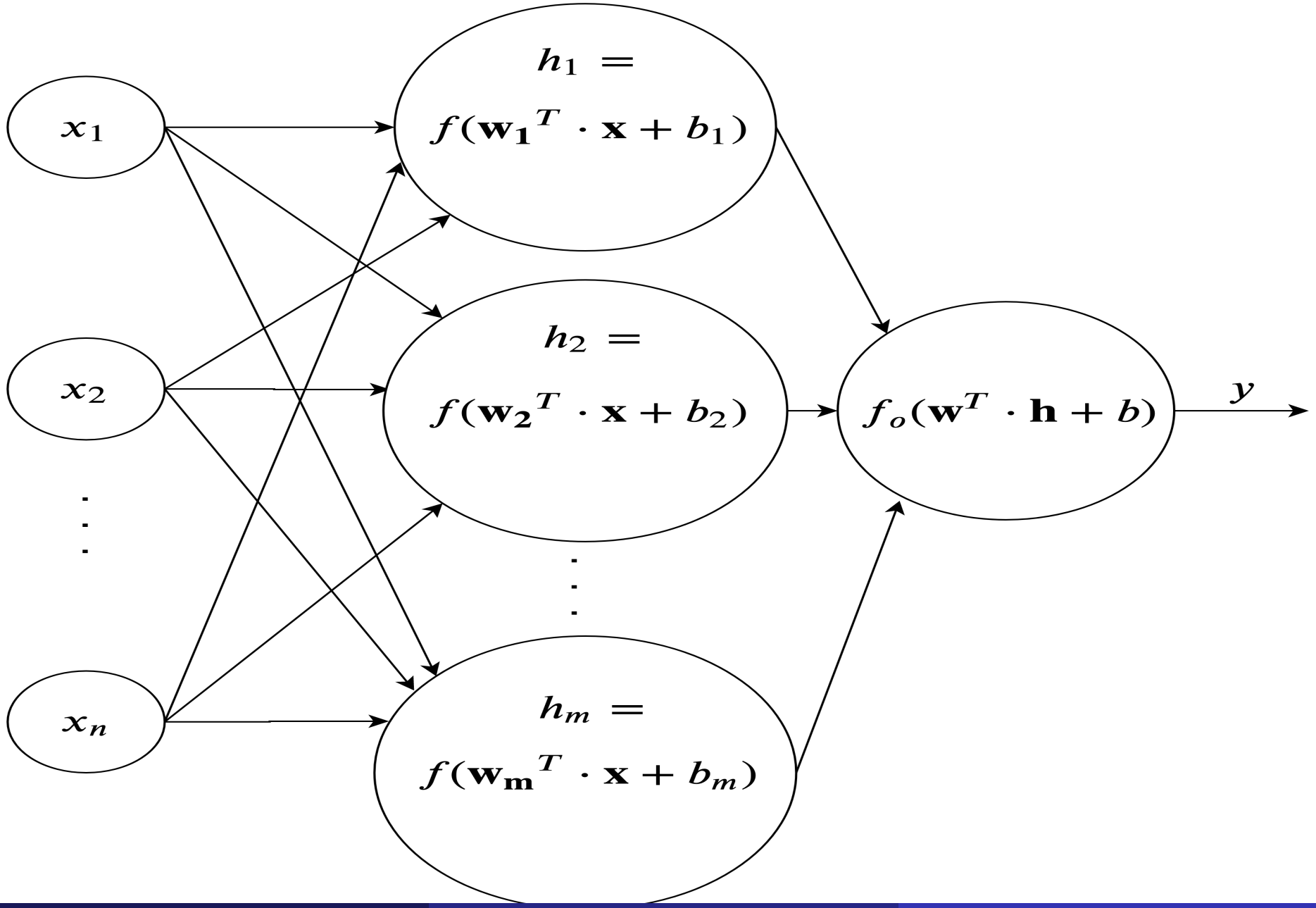


- $y = \sigma(\mathbf{w}^T \cdot \mathbf{x}) = \dfrac{1}{1 + \exp(-\mathbf{w}^T \cdot \mathbf{x})}$
- оптимизируя функцию потерь методом градиентного спуска, находим параметры $\mathbf{w}$

Идея

Перед формированием ответа модели y можно скомбинировать и нелинейно преобразовать исходные признаки $\mathbf{x}$, причем параметры этой трансформации должны быть обучаемыми

Персептрон с одним скрытым слоем



Персептрон

- Скрытый слой можно записать как $\mathbf{h} = f(\mathbf{W}\mathbf{x} + \mathbf{b})$; $\mathbf{W} \in \mathbb{R}^{m \times n}$; $\mathbf{x} \in \mathbb{R}^n$; $\mathbf{b}, \mathbf{h} \in \mathbb{R}^m$
- На практике используется не более 2-3 скрытых слоев. Больше параметров – выше вероятность переобучения

Персептрон

$y = f_o(\mathbf{h})$ – выходной слой. В качестве f_o обычно используется линейная модель:

- $y = \mathbf{w}^T \cdot \mathbf{h} \in (-\infty, +\infty)$ – регрессия
- $y = \sigma(\mathbf{w}^T \cdot \mathbf{h}) = \dfrac{1}{1 + \exp -(\mathbf{w}^T \cdot \mathbf{h})} \in (0, 1)$ – бинарная классификация

- $\mathbf{y} = softmax(\mathbf{W}\mathbf{h}) \in (0, 1)^c, \mathbf{W} \in \mathbb{R}^{c \times m}$ – классификация на $c > 2$ классов

$$softmax_i(\mathbf{x}) = \frac{exp(x_i)}{\sum_{k=1}^{n} exp(x_k)}$$

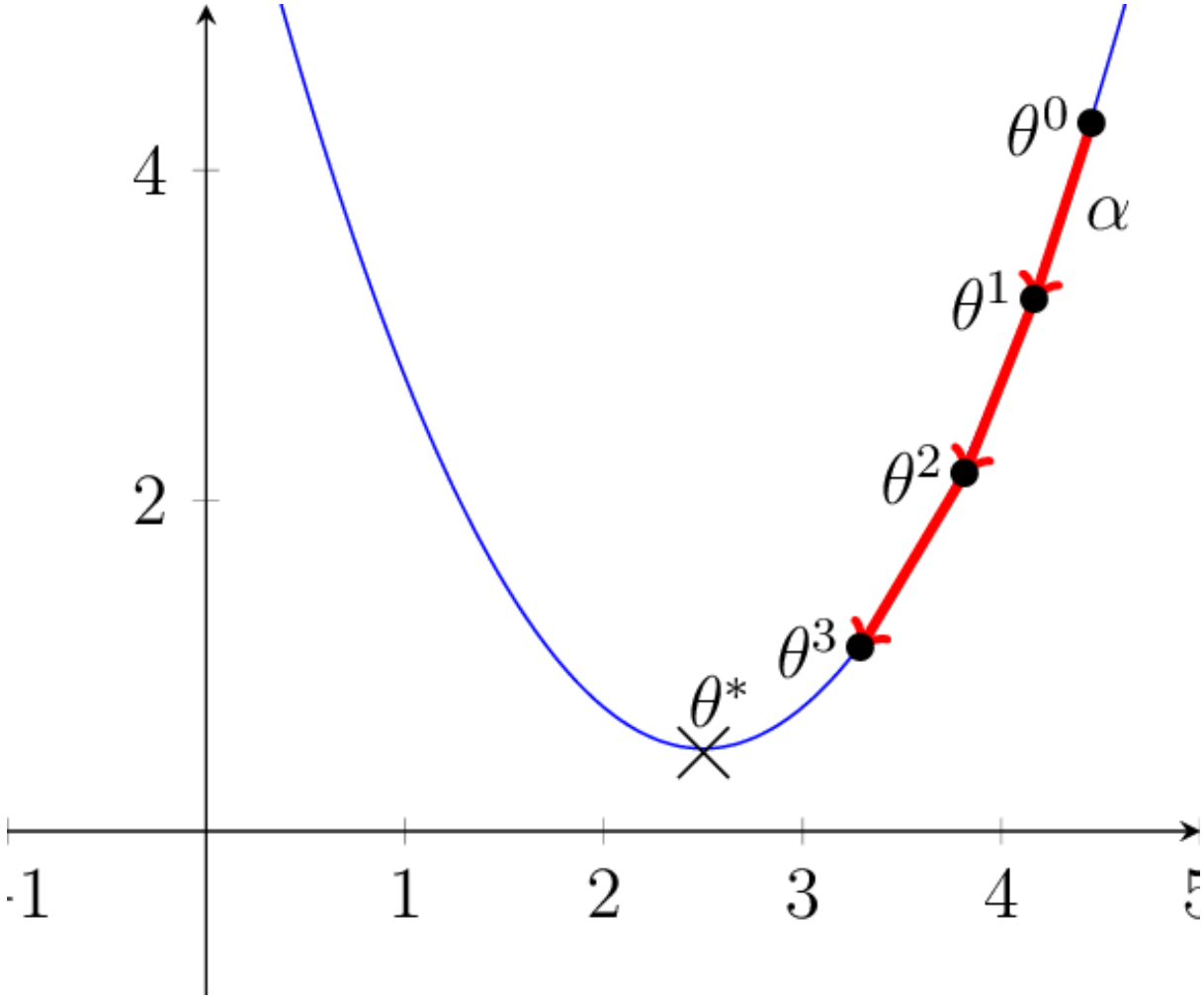
Обучение нейронной сети с учителем

- Пусть $(x_i, \tilde{y}_i), i \in [1, n]$ – обучающая выборка, где x_i – объекты, $\tilde{y}_i$ – правильные ответы
- Пусть $y = y_\theta(x)$ – дифференцируемая нейронная сеть с параметрами θ. Процесс обучения заключается в нахождении оптимальных параметров
- Параметры θ должны быть такими, чтобы нейронная сеть выдавала “близкие” к правильным ответам на элементах обучающей выборки

Обучение нейронной сети с учителем

- Для оценки погрешности алгоритма выбирается функция ошибки $l = l(y, \tilde{y}), l \in \mathbb{R}$. К примеру, из раннее рассмотренных:
  - MSE (линейная регрессия): $l = (y - \tilde{y})^2$
  - cross-entropy (логистическая регрессия): $l = -(\tilde{y} \log y + (1 - \tilde{y}) \log(1 - y))$
- По всей обучающей выборке строим функционал качества, применяя нейронную сеть и сравнивая ее ответы с правильными:
$$L = L(\theta) = \frac{1}{n} \sum_{i=1}^{n} l(y_\theta(x_i), \tilde{y}_i) = \frac{1}{n} \sum_{i=1}^{n} l_i$$
- Т.к. L – дифференцируемая функция, то можем пытаться попасть в ее локальный минимум, оптимизируя параметры θ с помощью градиентного спуска: $\theta^{i+1} = \theta^i - \alpha \cdot \nabla L(\theta^i)$

Обучение нейронной сети с учителем



Несколько шагов градиентного спуска

SGD

- Вычисление L при больших n – ресурсоемкая задача
- Stochastic gradient descent (SGD) – на каждом шаге градиентного спуска случайно выбираем **один** пример j из обучающей выборки и обновляем параметры: $\theta^{i+1} = \theta^i - \alpha \cdot \nabla l_j(\theta^i)$
- SGD – подвержен “выбросам”, поэтому шаг α для сходимости следует выбирать маленьким, что приводит к большому числу шагов, требуемых для сходимости
- Вычисление ошибки не параллелится по элементам обучающей выборки

miniBatch SGD

- miniBatch SGD – на каждом шаге градиентного спуска случайно выбираем **m** примеров ($1 < m << n$) из обучающей выборки и обновляем параметры: $\theta^{i+1} = \theta^i - \alpha \cdot \nabla[\frac{1}{m}\sum_{j=1}^{m} l_j(\theta^i)]$
- Более устойчив к “выбросам”, поэтому можем выбирать больший шаг α
- Ошибку $\sum_{j=1}^{m} l_j(\theta^i)$ можно считать параллельно

Обучение

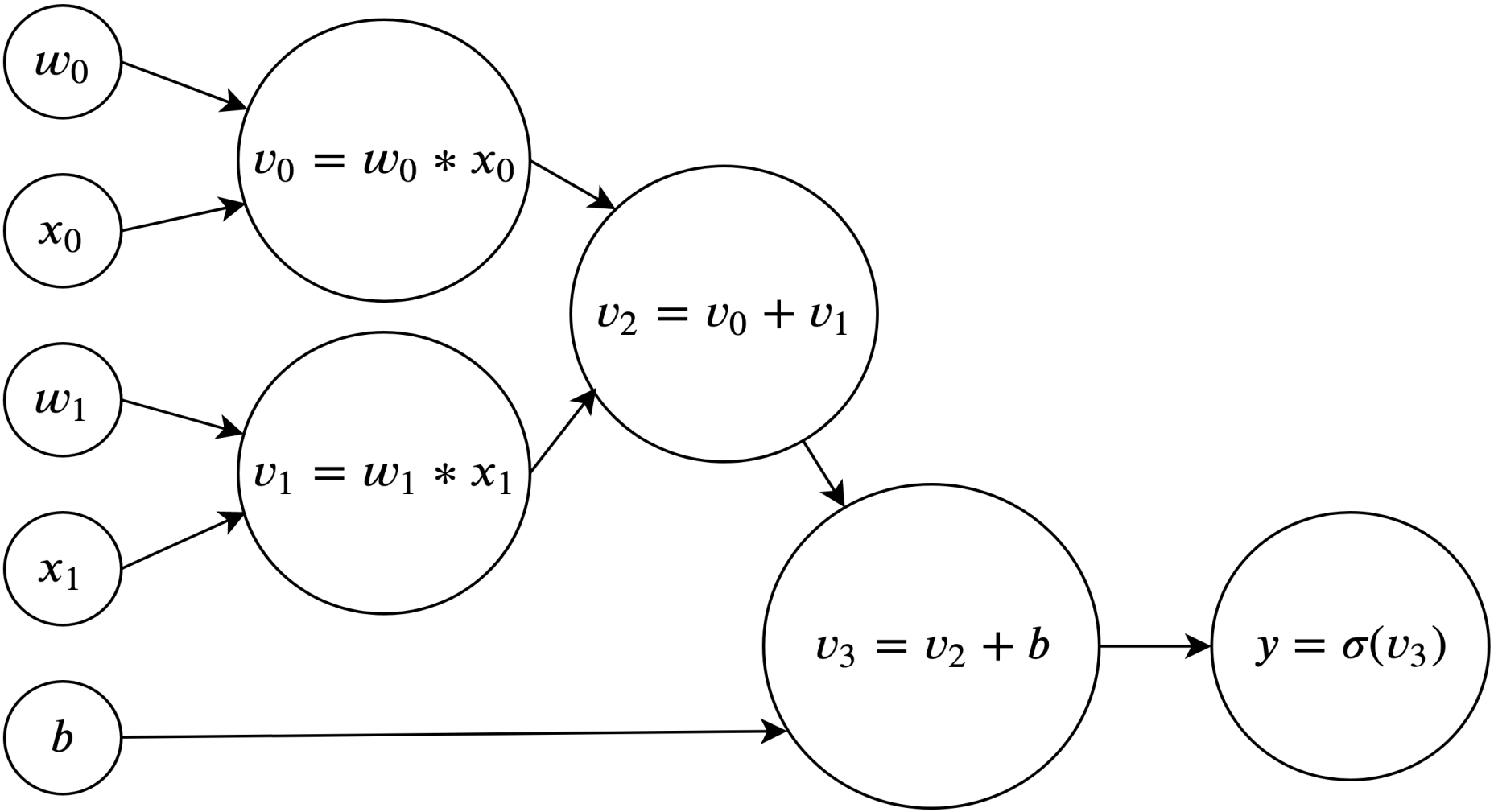
Обучение проводят некоторое заданное число “эпох” – полных итераций по обучающей выборке и останавливаются, когда качество на валидационной выборке перестает улучшаться.
В настоящее время, в основном, используются адаптивные методы градиентного спуска, которые задают значение α на каждом шаге в зависимости от предыдущих значений градиента – Momentum, RMSProp, Adam и т.д.

Вопрос

Как вычислять градиент $\nabla L(\theta^i)$ по каждому параметру θ^i_j глубокой нейронной сети (в общем случае, сложной дифференцируемой функции)?

Граф вычислений

Логистическая регрессия: $y = \frac{1}{1+\exp -(w_0 \cdot x_0 + w_1 \cdot x_1 + b)}$



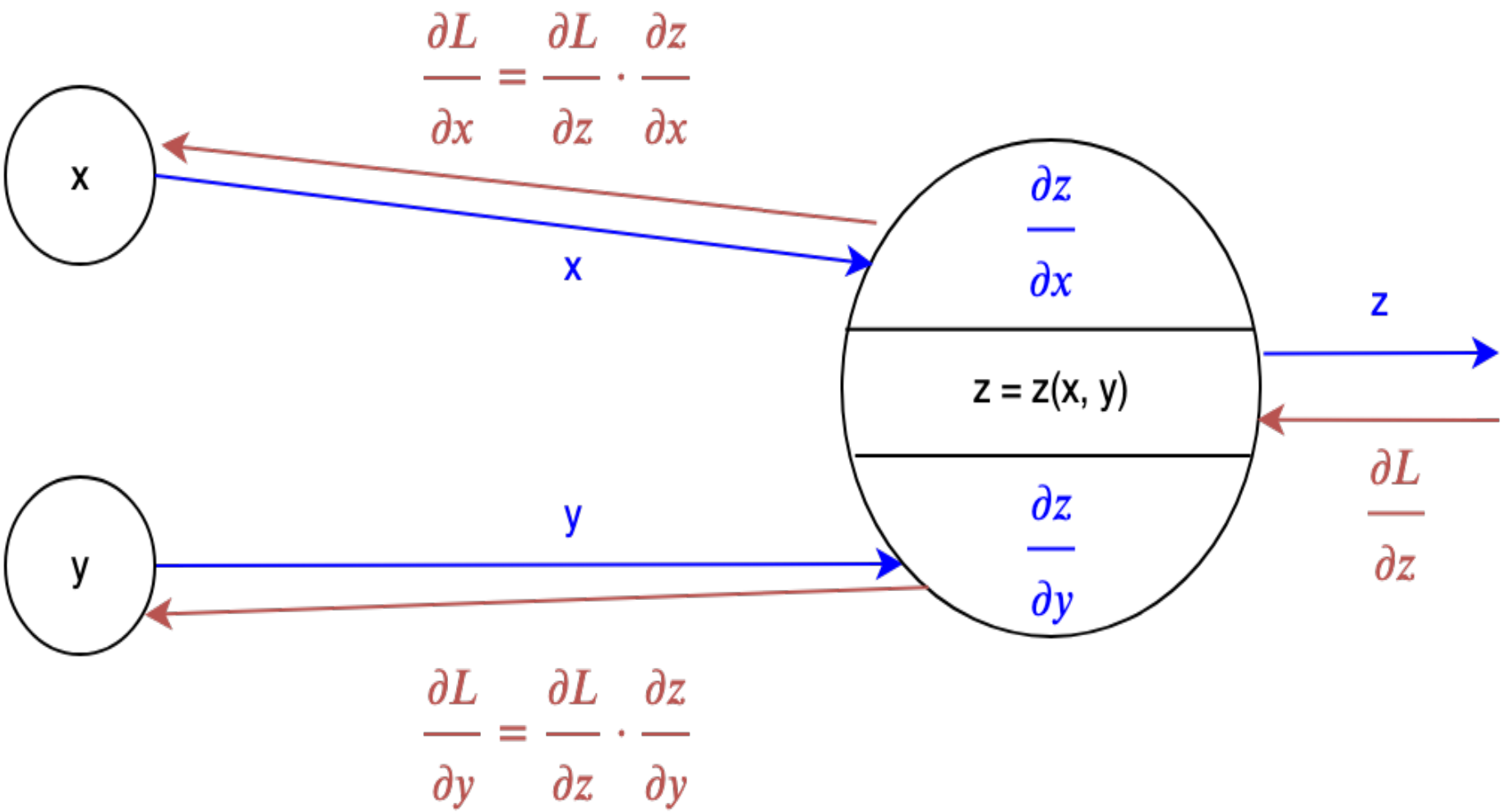
Backpropagation

Backpropagation

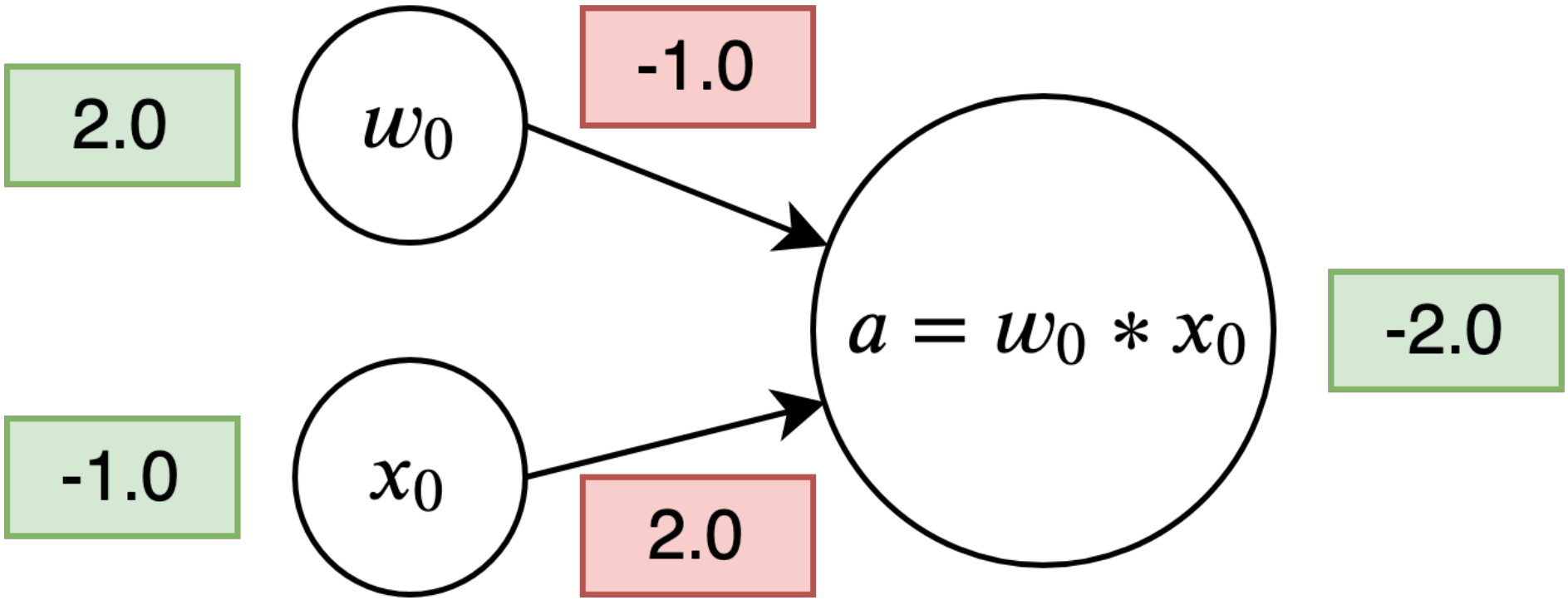
На входе граф вычислений некоторой дифференцируемой функции y.
Алгоритм:

1. Вычисляем значения и частные производные для каждого из узлов (forward pass)
2. Вычисляем частные производные реализуемой функции y по каждому из ее аргументов – узлов графа (backward pass)

Backpropagation



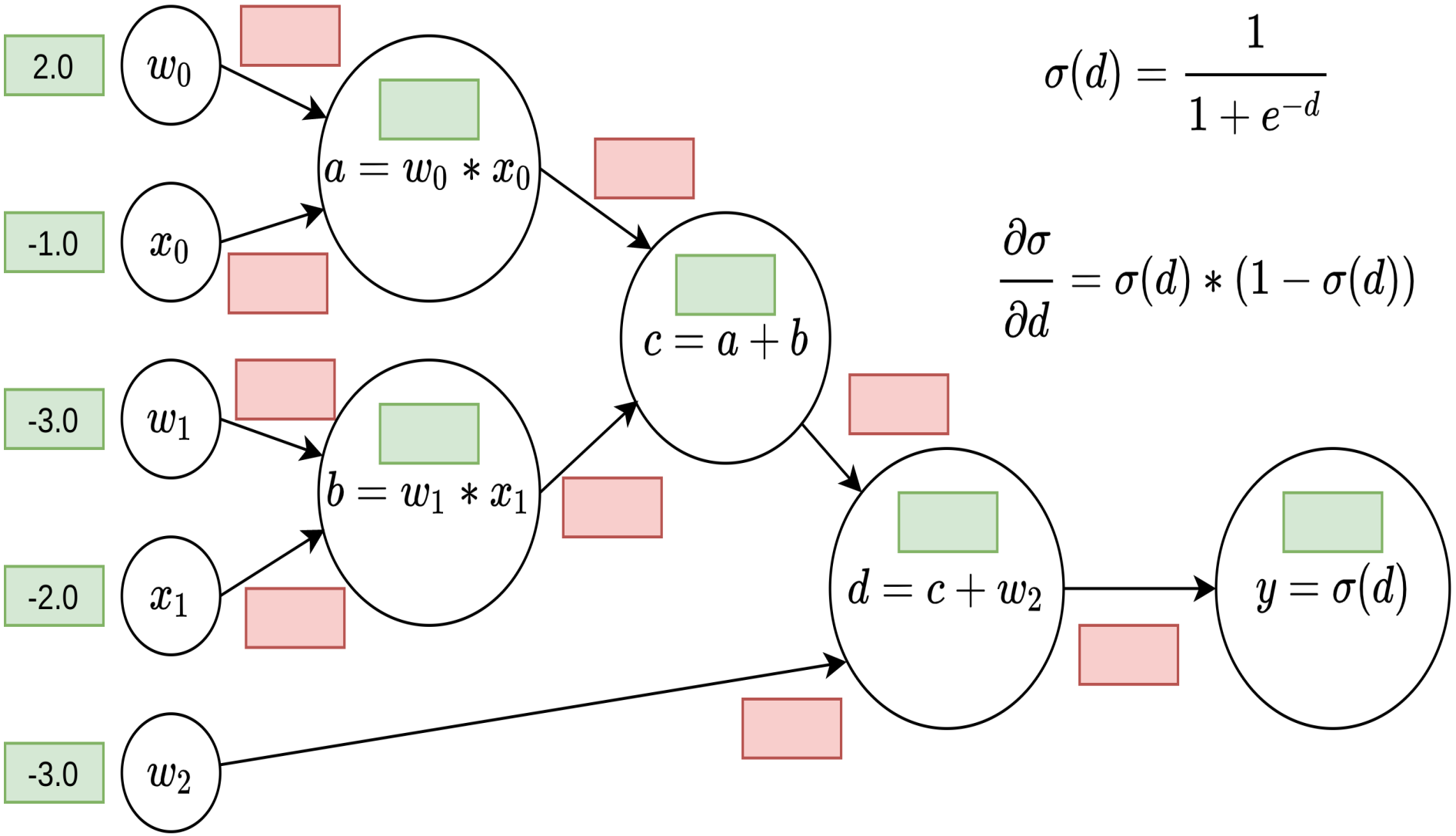
Forward pass



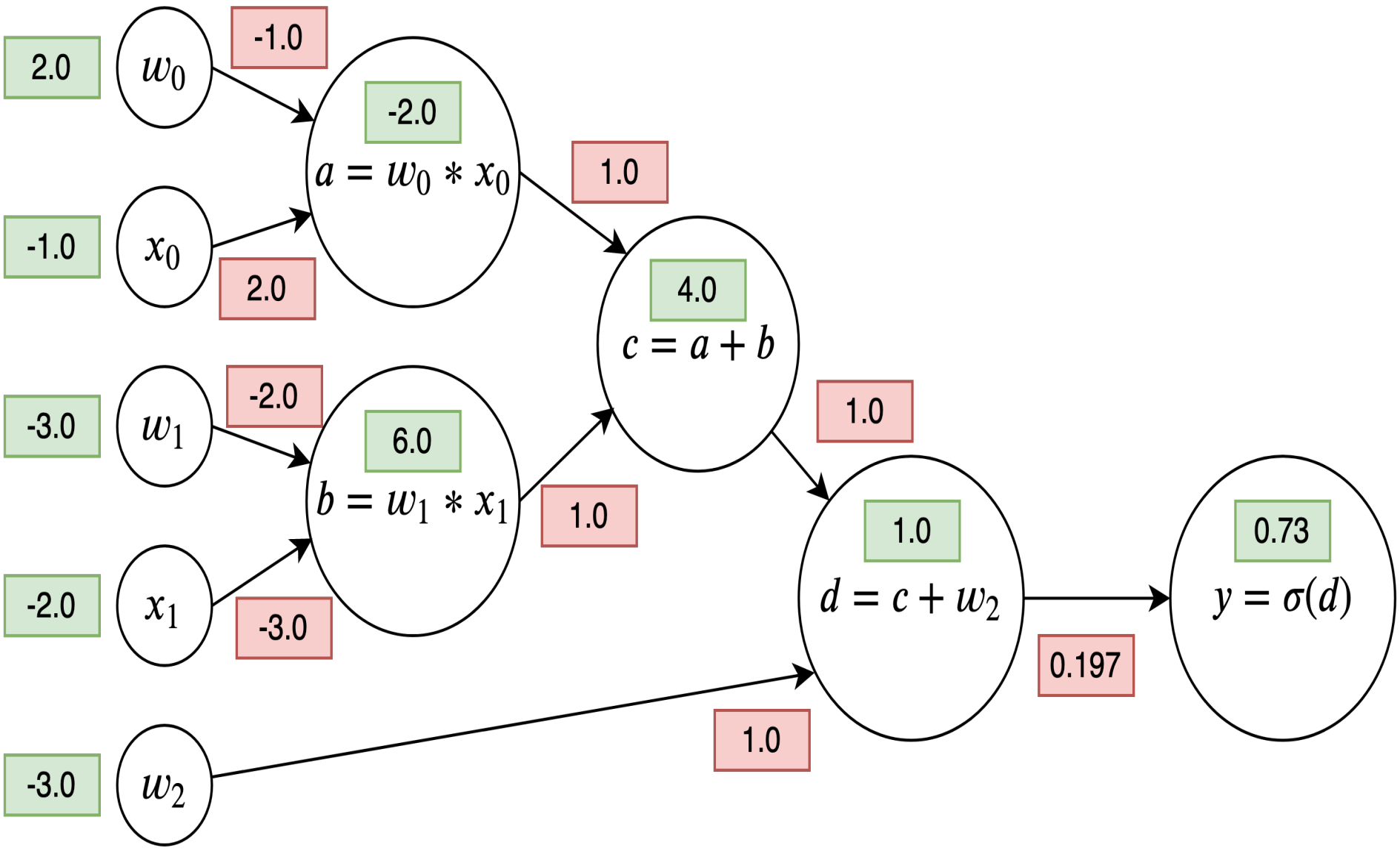
зеленый цвет – значение функции

красный цвет – значение частной производной

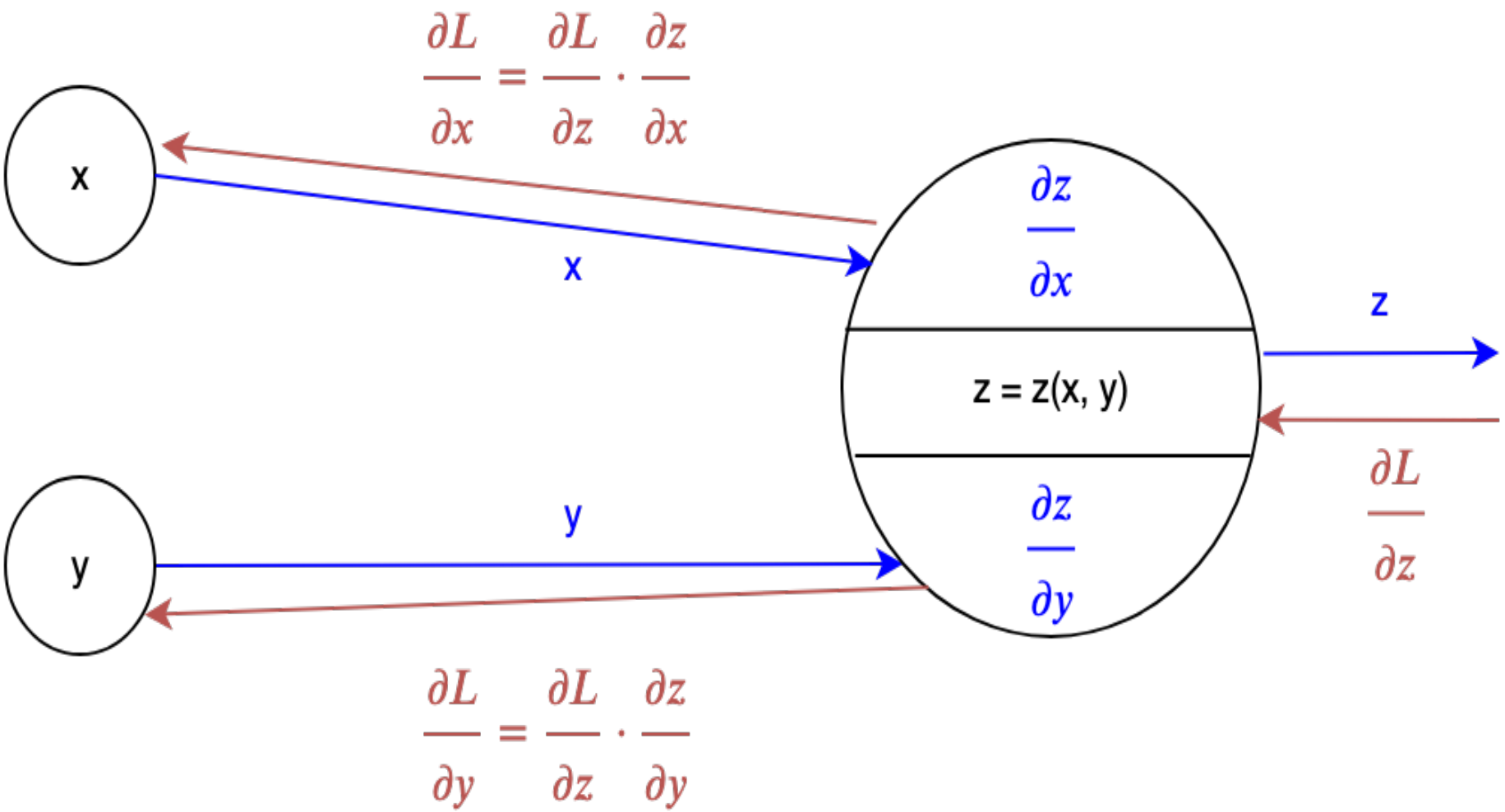
Пример forward pass



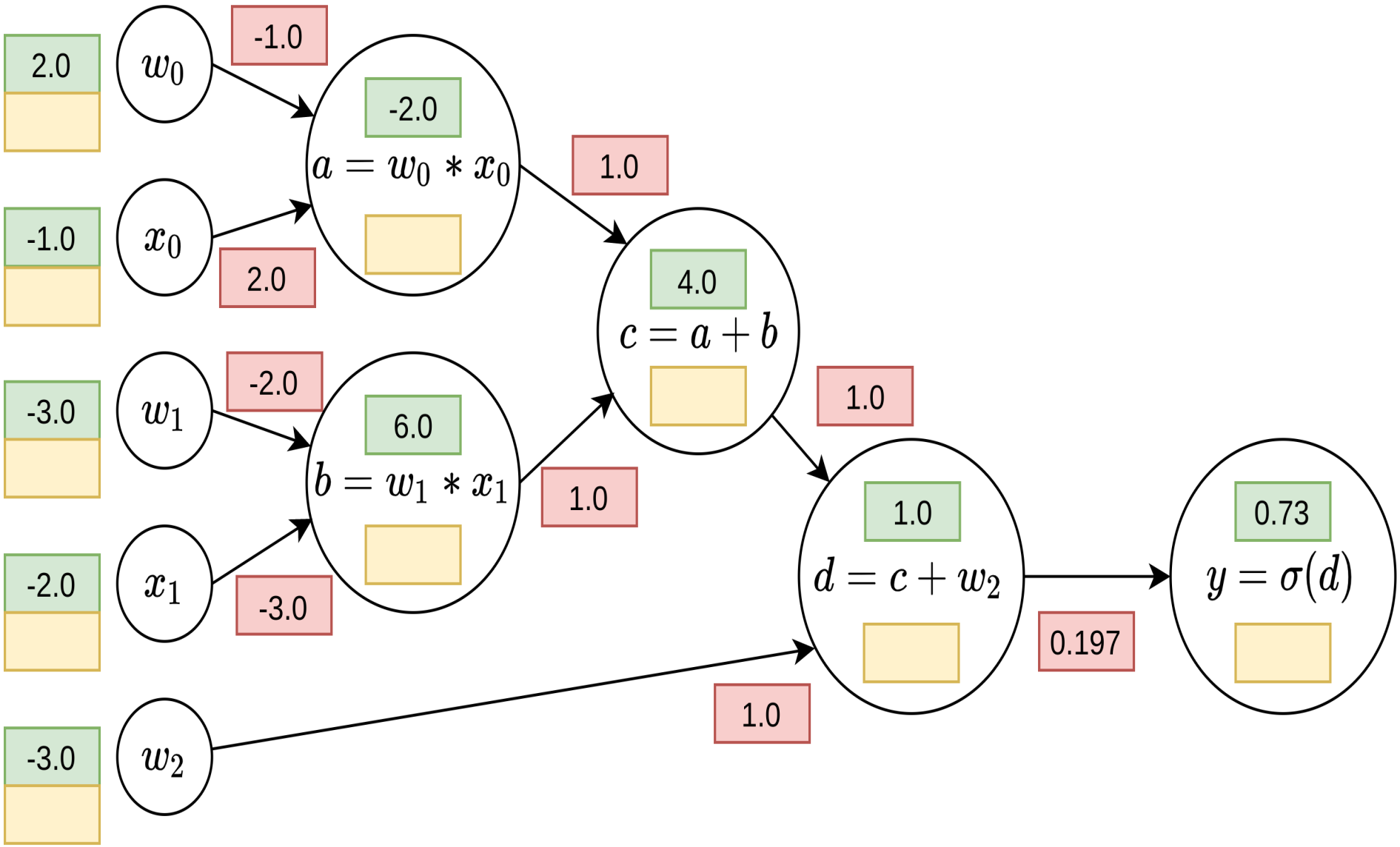
Пример forward pass



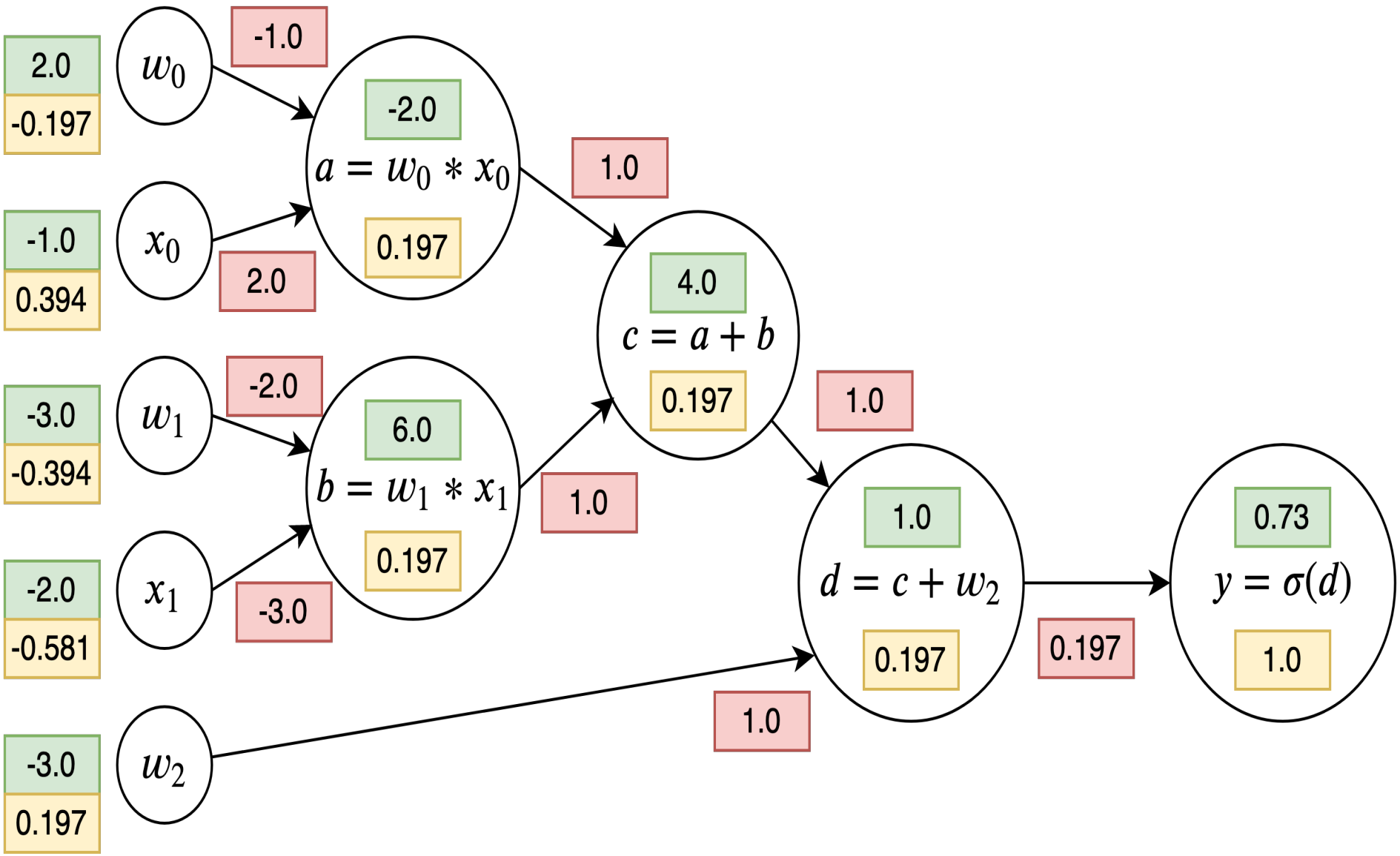
Backpropagation



Пример backward pass



Пример backward pass

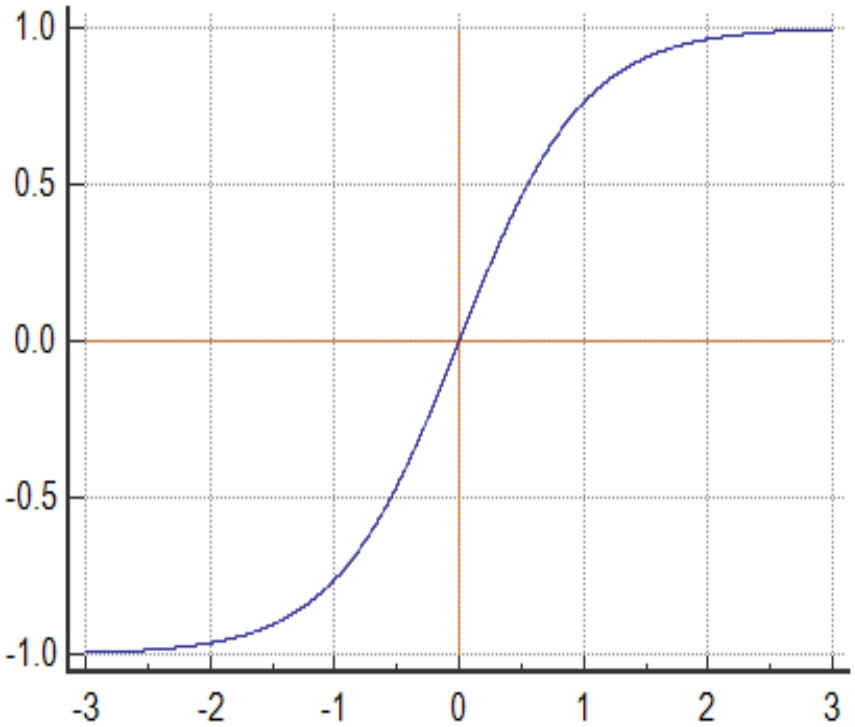


Проблема затухающих градиентов

При backpropagation в глубокой нейронной сети градиент на первых слоях может быть очень маленьким из-за последовательного умножения на $|\frac{\partial z}{\partial y}| << 1$, поэтому эти слои будут обновляться “слабо”.

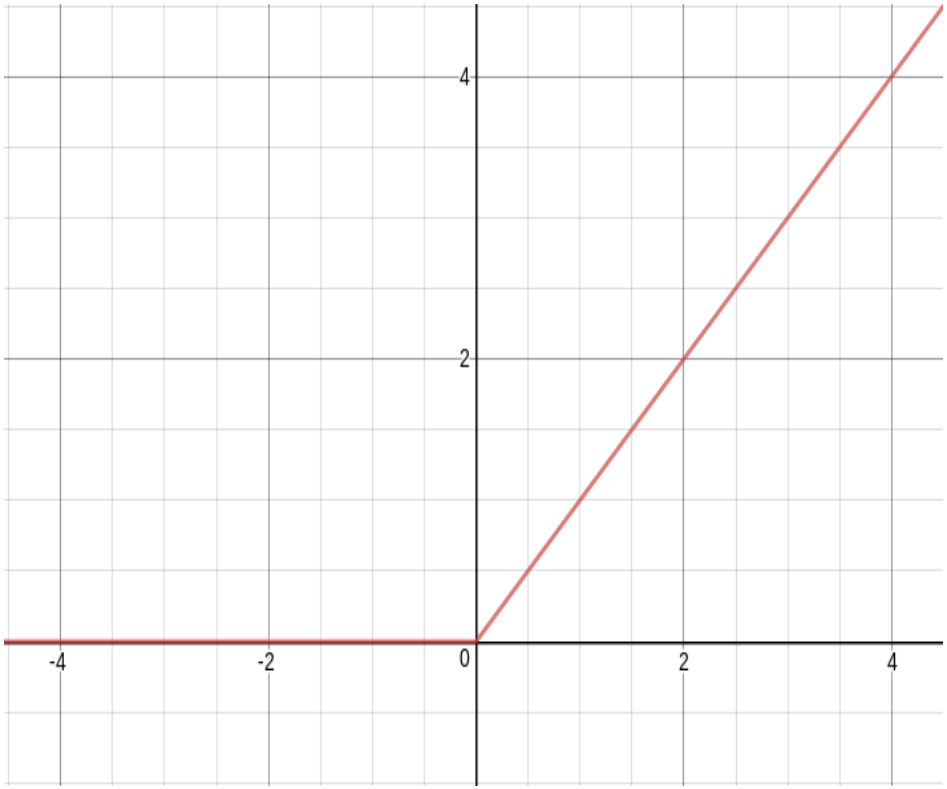
Чтобы с этим бороться, выбирают специальные функции активации и разрабатывают специальные архитектуры, распространяющие градиент на первые слои (skip / residual connections).

Функции активации – tanh



- $y = \tanh(x)$
- $y \in [-1, 1]$
- $\frac{\partial y}{\partial x} \leq 1.0$

Функции активации – ReLU



- $y = ReLU(x) = \max(0, x)$
- $y \in [0, +\infty)$
- $\frac{\partial y}{\partial x} \in \{0, 1\}$
- эффективное вычисление
- Часто используется в скрытых слоях

Начальная инициализация

Перед обучением нейронной сети нужно проинициализировать ее параметры θ. Обычно используется случайная инициализация вещественными числами из интервала (-1, 1).

Почему все веса нельзя инициализировать 0? Константой?

Модельная задача

Будем классифицировать СМС на два класса: не спам, спам

Nah I don't think he goes to usf, he lives around here though – не спам

You are a winner U have been specially selected 2 receive £1000 or a 4* holiday (flights inc) speak to a live operator 2 claim 0871277810910p/min (18+) – спам

Постановка задачи

Построить признаковое пространство X и модель $y : X \to \{0, 1\}$

Модельная задача

Прежде, чем приступать к решению задачи, нужно понять, как оценивать качество разработанного метода.

Важно

Метрика должна отражать реальное качество, а не быть “высокой цифрой“, даже если задача решена плохо

Метрики качества бинарной классификации

Типы ошибок

- tp – true positive, метод – 1, gold метка – 1
- tn – true negative, метод – 0, gold метка – 0
- fp – false positive, метод – 1, gold метка – 0
- fn – false negative, метод – 0, gold метка – 1

- $accuracy = \frac{tp + tn}{tp + tn + fp + fn}$ – плоха при несбалансированности классов
- $precision = \frac{tp}{tp + fp}$, $recall = \frac{tp}{tp + fn}$, $F1 = 2 * \frac{precision * recall}{precision + recall}$
- Можно AUC-ROC[a]

[a] https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

Построение признакового пространства

- Провести токенизацию – разбить текст на словоформы (токены)
- Построить признаковый вектор для текста, как последовательности токенов

BoW признаки

BoW – bag of words (мешок слов)

- Пусть V – словарь, состоящий из всех слов текстов обучающей выборки. Перенумеруем его.
- Тогда каждое слово $t \in V$ можно представить вектором v размерности $|V|$, где $v_i = 1$, i – номер t в словаре, иначе $v_j = 0$. Такая кодировка называется one-hot кодировкой.
- Теперь текст можно представить вектором, как суммой one-hot векторов слов, из которых он состоит. На каждой позиции такого вектора стоит количество раз, сколько слово с номером i входит в текст.

Фреймворки для python

- sklearn.neural_network – недостаточная конфигурация
- tensorflow – высокий порог вхождения
- keras, pytorch
- Для pytorch есть фреймворки, упрощающие разработку training-loop – pytorch-lightning, ignite.

Однослойный персептрон на pytorch

```
import torch

class BinaryClassificationPerceptron(torch.nn.Module):
    def __init__(self, input_size: int, hidden_size: int):
        super().__init__()

        self._hidden = torch.nn.Linear(
            input_size, hidden_size)
        self._activation = torch.nn.ReLU()

        self._out_layer = torch.nn.Linear(hidden_size, 1)
        self._out_sigmoid = torch.nn.Sigmoid()

    def forward(self, sample):
        sample = self._activation(self._hidden(sample))
        return self._out_sigmoid(self._out_layer(sample))
```

Разметка последовательности – NERC

[**Александр Пушкин** | ЛИЧ] родился в [**Москве** | ЛОК]

IO-кодирование:

I-ЛИЧ I-ЛИЧ O O I-ЛОК

Кодирование с учетом контекста

Текст можно представить как последовательность векторов словоформ $\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_l}$, $\mathbf{x_i} \in \mathbb{R}^d$.

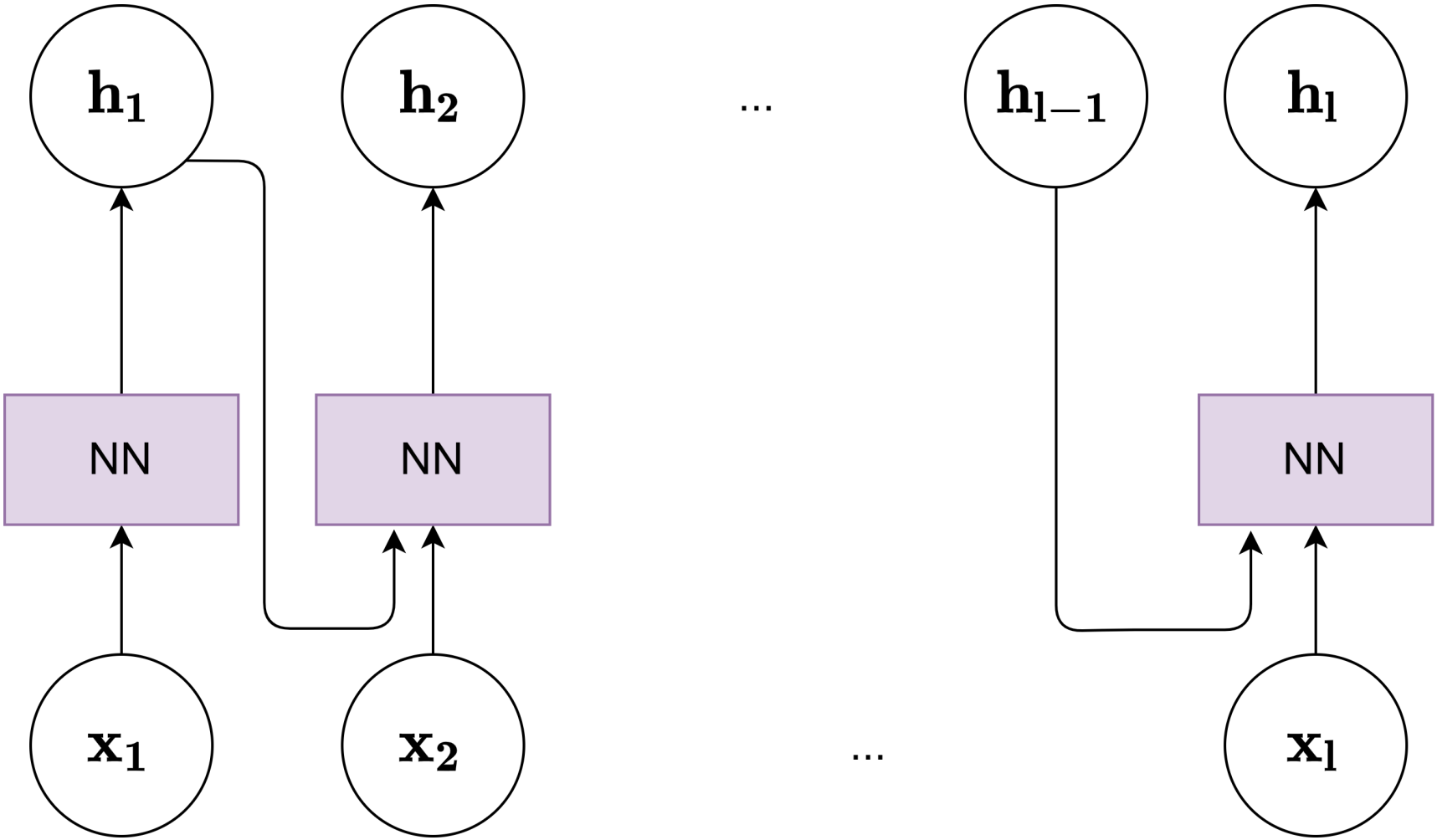
Как получить представление каждого слова, с учетом его текущего контекста, чтобы классифицировать его на предмет вхождения в именованную сущность? Нужно учитывать, что длина последовательности не является постоянной величиной.

Кодирование с учетом контекста

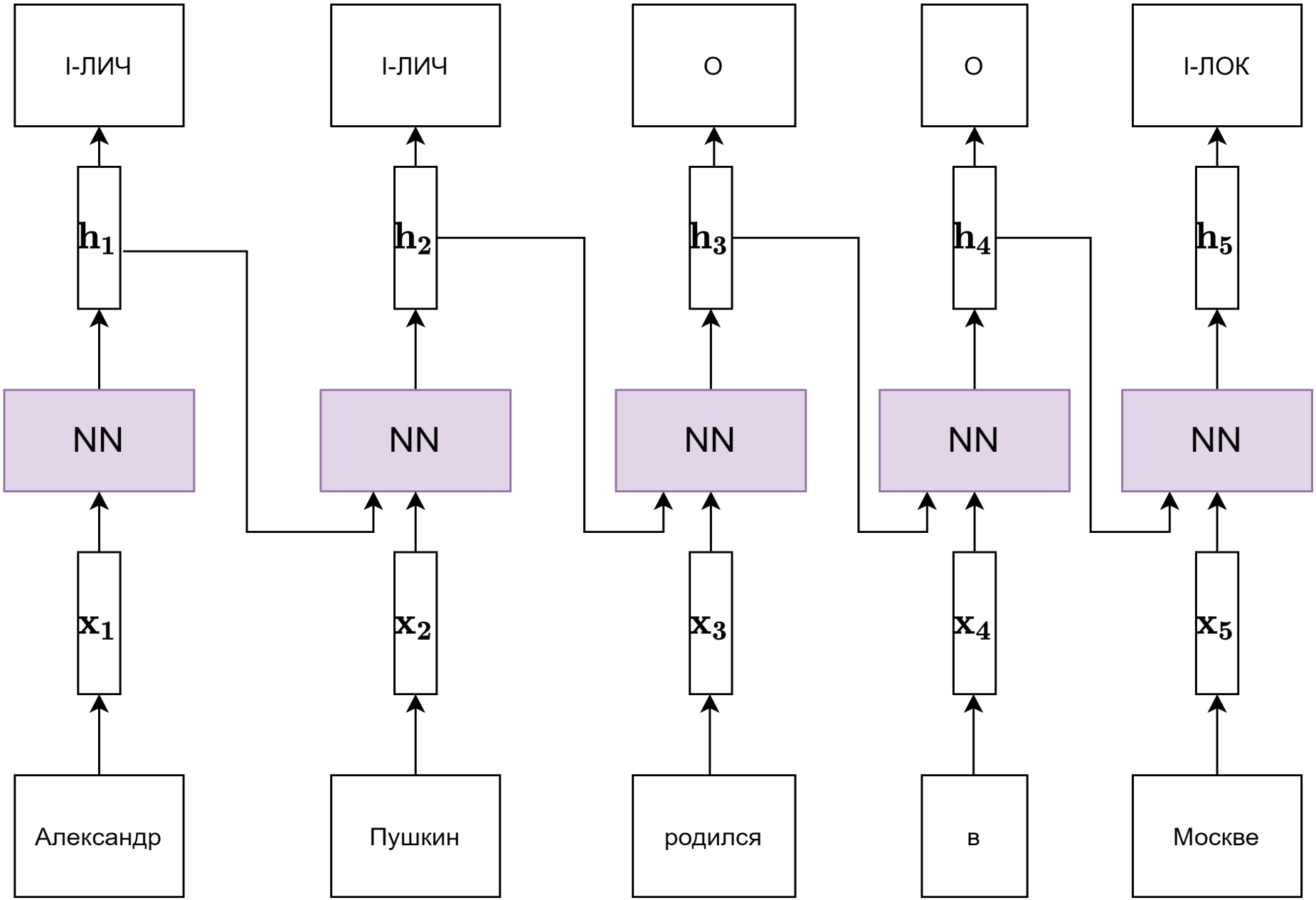
- сверточные нейронные сети (CNN)
- нейронные сети на основе механизма внимания (Transformer)
- **рекуррентные нейронные сети (GRU, LSTM)**

Рекуррентные нейронные сети

Формально: $\mathbf{h_t} = f(\mathbf{x_t}, \mathbf{h_{t-1}}), \mathbf{h_0} = \mathbf{0}, t \in [1, l]$



Рекуррентные нейронные сети



Простейшая рекуррентная нейронная сеть

- $\mathbf{h_0} = \mathbf{0}$
- $\mathbf{h_t} = f(\mathbf{W_x x_t} + \mathbf{W_h h_{t-1}} + \mathbf{b})$

Проблема catastrophic forgetting

- Пусть на вход дано 3 вектора $\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}$
- Развернем выход сети при $\mathbf{b} = \mathbf{0}$:

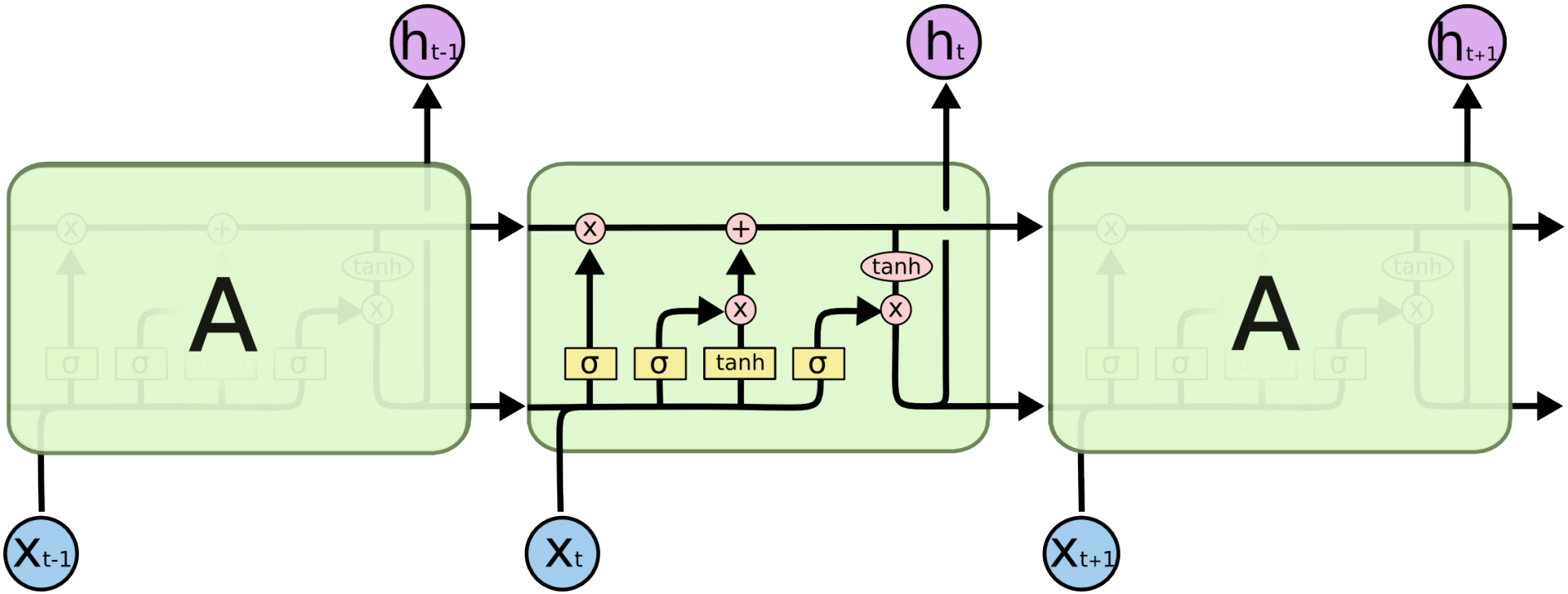
$$\mathbf{h_3} = f(\mathbf{W_x x_3} + \mathbf{W_h} f(\mathbf{W_x x_2} + \mathbf{W_h h_1}))$$

- Считая, что мы работаем со скалярами:

$$\frac{\partial h_3}{\partial h_1} = f'(W_x x_3 + W_h f(W_x x_2 + W_h h_1)) * W_h * f'(W_x x_2 + W_h h_1) * W_h$$

- Если $|f'| < 1$, то $|\frac{\partial h_3}{\partial h_1}| << 1$. Это означает, что предыдущий выход не влияет на текущее решение сети. А значит нейронная сеть забывает, что было раннее :(

LSTM



[0] https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM

- $\odot$ – поэлементное произведение векторов (произведение Адамара)
- $\mathbf{f_t} = \sigma(\mathbf{W_f x_t} + \mathbf{U_f h_{t-1}} + \mathbf{b_f})$ – что забываем
  $\mathbf{i_t} = \sigma(\mathbf{W_i x_t} + \mathbf{U_i h_{t-1}} + \mathbf{b_i})$ – что запоминаем
  $\mathbf{o_t} = \sigma(\mathbf{W_o x_t} + \mathbf{U_o h_{t-1}} + \mathbf{b_o})$ – что выдаем
  $\mathbf{\tilde{c}_t} = \tanh(\mathbf{W_c x_t} + \mathbf{U_c h_{t-1}} + \mathbf{b_c})$
  $\mathbf{c_t} = \mathbf{f_t} \odot \mathbf{c_{t-1}} + \mathbf{i_t} \odot \mathbf{\tilde{c}_t}$
  $\mathbf{h_t} = \mathbf{o_t} \odot \mathbf{\tanh}(\mathbf{c_t})$
- $\dfrac{\partial \mathbf{c_t}}{\partial \mathbf{c_{t-1}}} = \mathit{diag}(\mathbf{f_t})$ – когда сеть не хочет забывать, градиент не затухает

BiLSTM

В задачах обработки текстов важен не только левый контекст словоформы, но и правый:

Александр Пушкин **родился** в Москве

Поэтому используют BiLSTM – две параллельные LSTM, первой на вход подается исходная последовательность, второй – исходная последовательность в обратном порядке. Их выходы $\overrightarrow{h_t}$ и $\overleftarrow{h_t}$ конкатенируются.

Агрегация закодированного контекста

Мы закодировали исходную последовательность векторов словоформ $\mathbf{x_1}, ..., \mathbf{x_l}$ в последовательность векторов $\mathbf{h_1}, ..., \mathbf{h_l}$, которые учитывают контекст.
Если мы хотим классифицировать текст, то нужно построить вектор $\mathbf{a}$ фиксированной размерности, который далее отправится в классификатор:

- взять последний выход $\mathbf{a} = \mathbf{h_l}$
- mean pooling $a_i = \frac{\sum_j h_{j_i}}{l}$
- max pooling $a_i = \max_j h_{j_i}$
- механизм внимания

Что посмотреть, если интересно

- A.Karpathy CS231n Winter 2016 lectures
- C.Manning CS224n Winter 2019 lectures
- Воронцов К.В. Машинное обучение

Вопросы?