Искусственные нейронные сети для обработки текстов

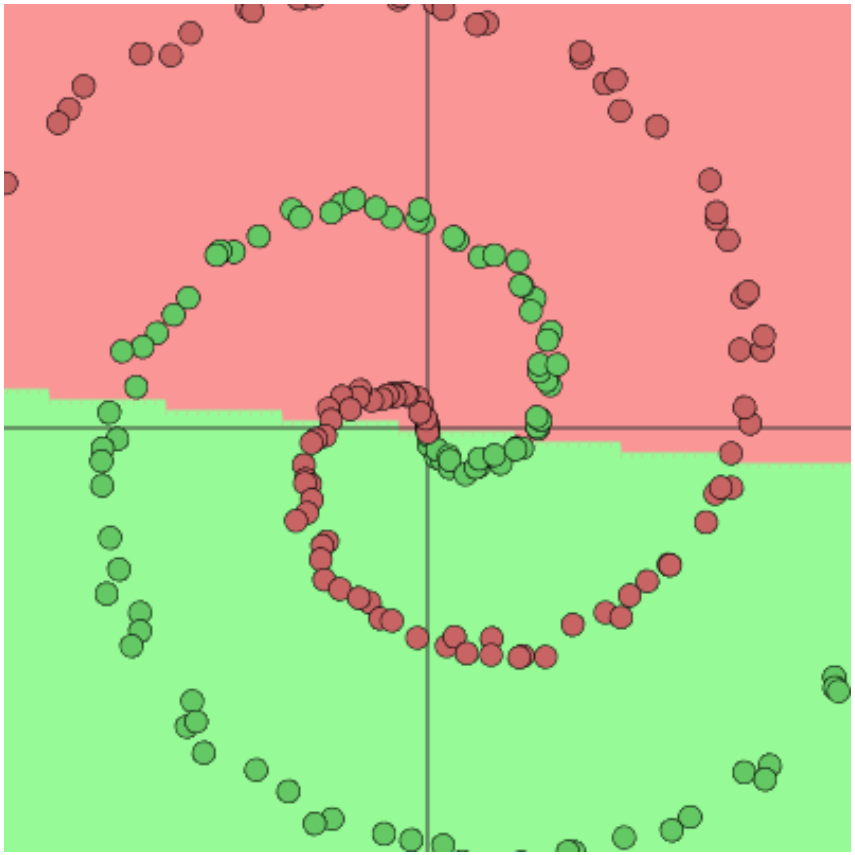
Трифонов Владислав

25 сентября 2019 г.

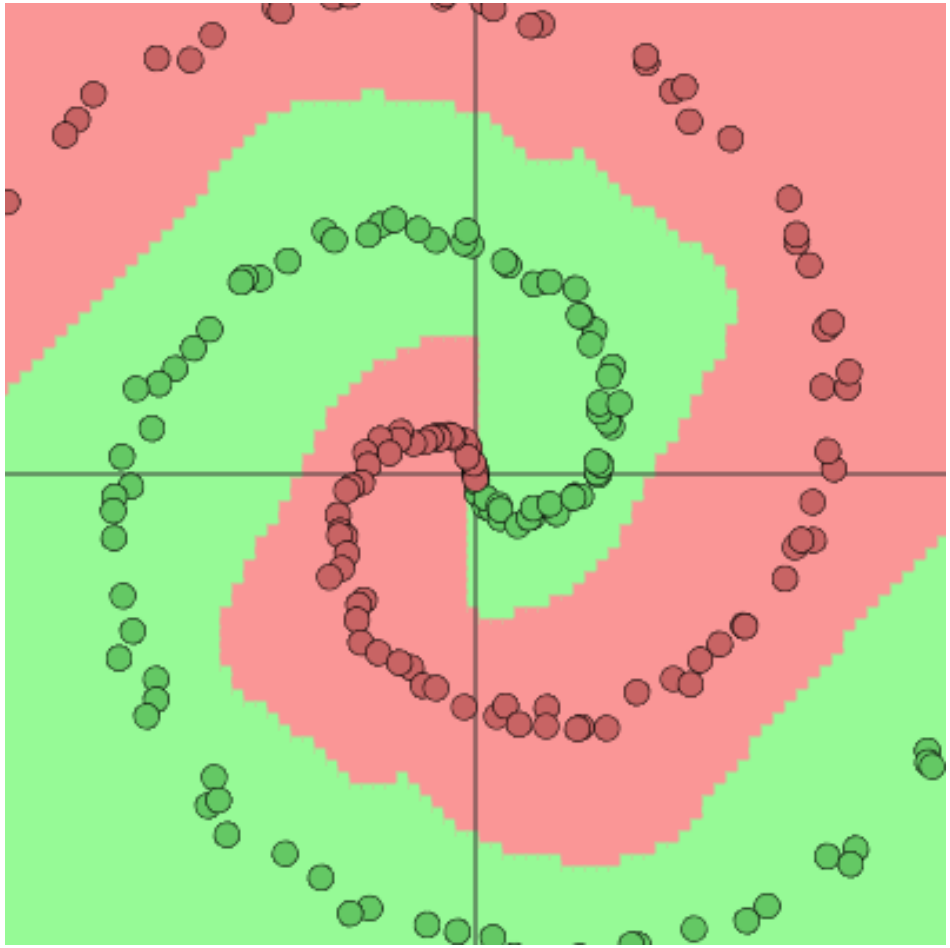
Введение

Нейронные сети – мощные алгоритмы машинного обучения, широко используемые при решении задач автоматической обработки текстов, изображений, видео и звуков.
Они могут находить сложные зависимости в данных, что недоступно для традиционных методов ML.

Введение



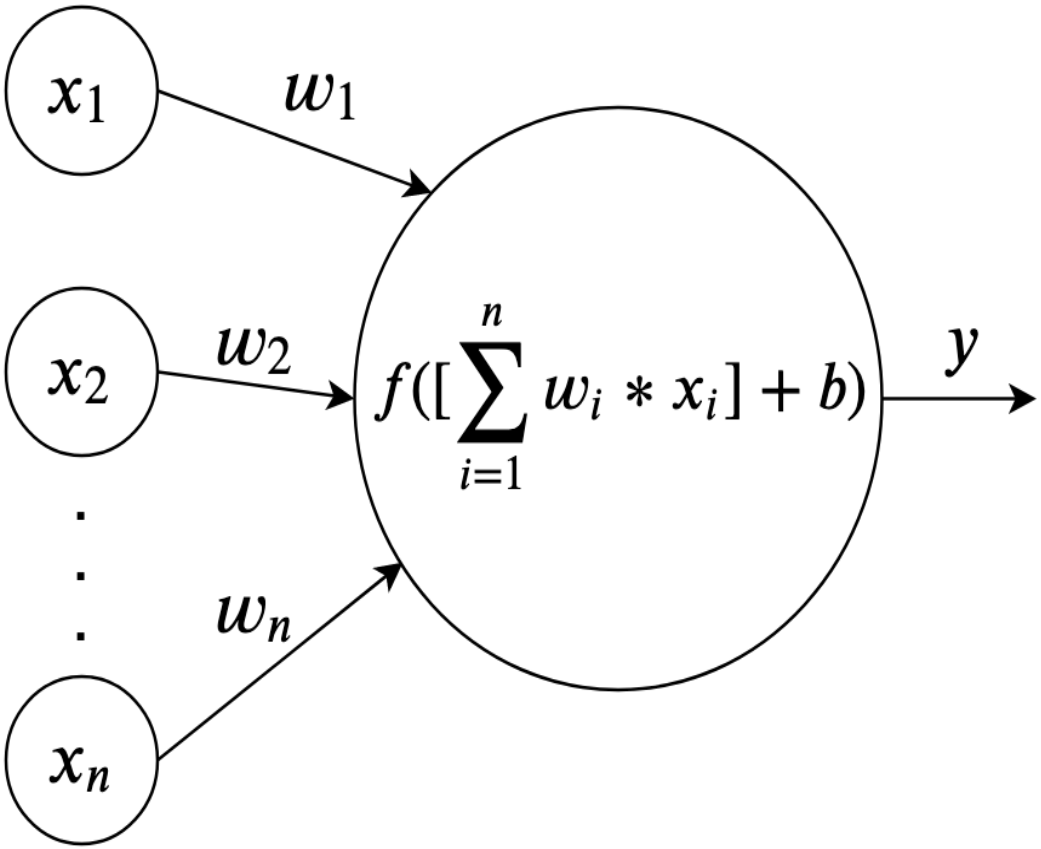
Линейный классификатор



Нейронная сеть с tanh скрытым слоем

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

Модель нейрона



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

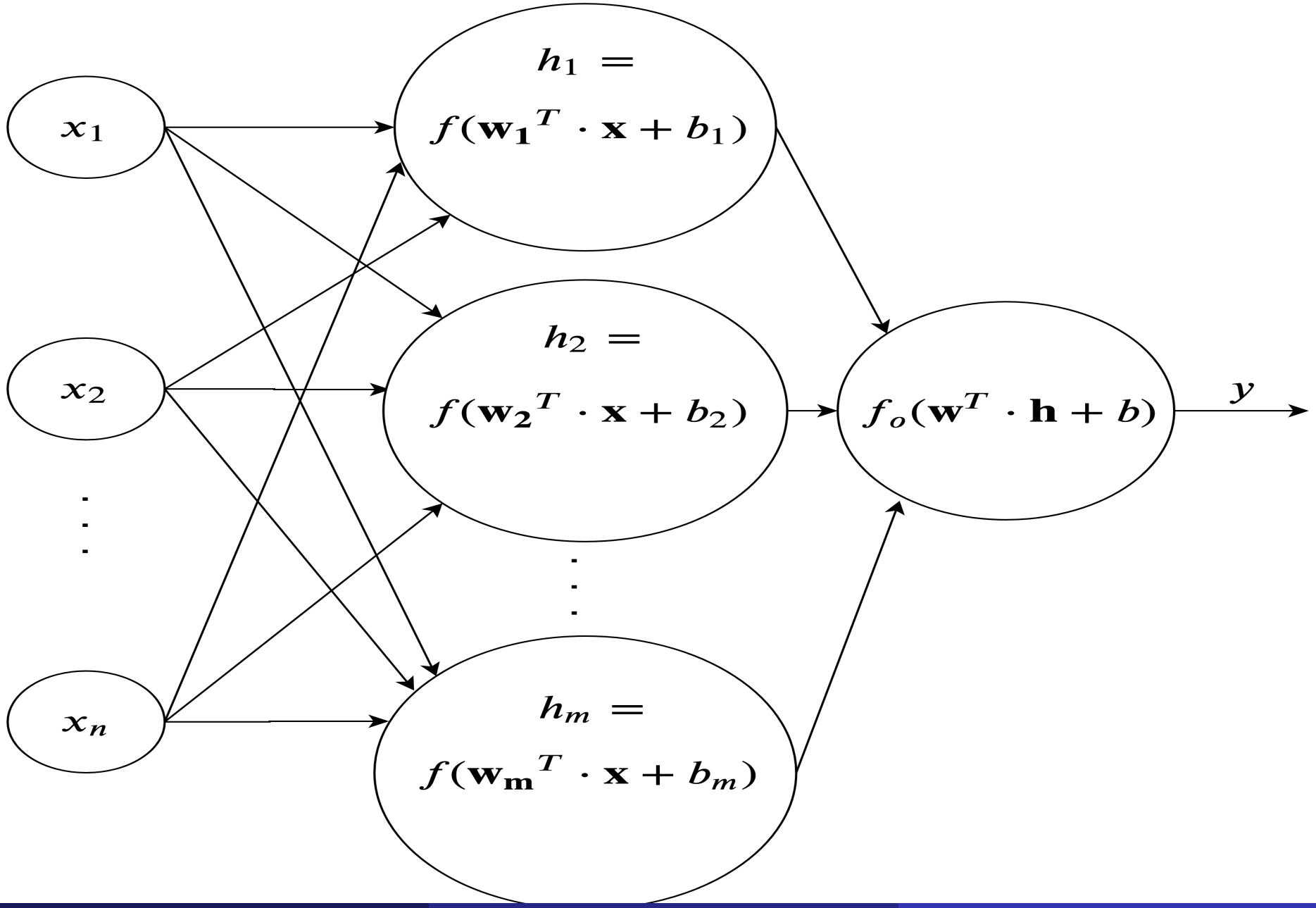
Модель нейрона

- Нейрон линейно комбинирует входные признаки и с помощью функции активации генерирует число – сигнал, который можно использовать как ответ классификатора или как новый признак
- Линейные модели (линейная регрессия, линейный классификатор, логистическая регрессия) – простейшие нейронные сети с одним нейроном

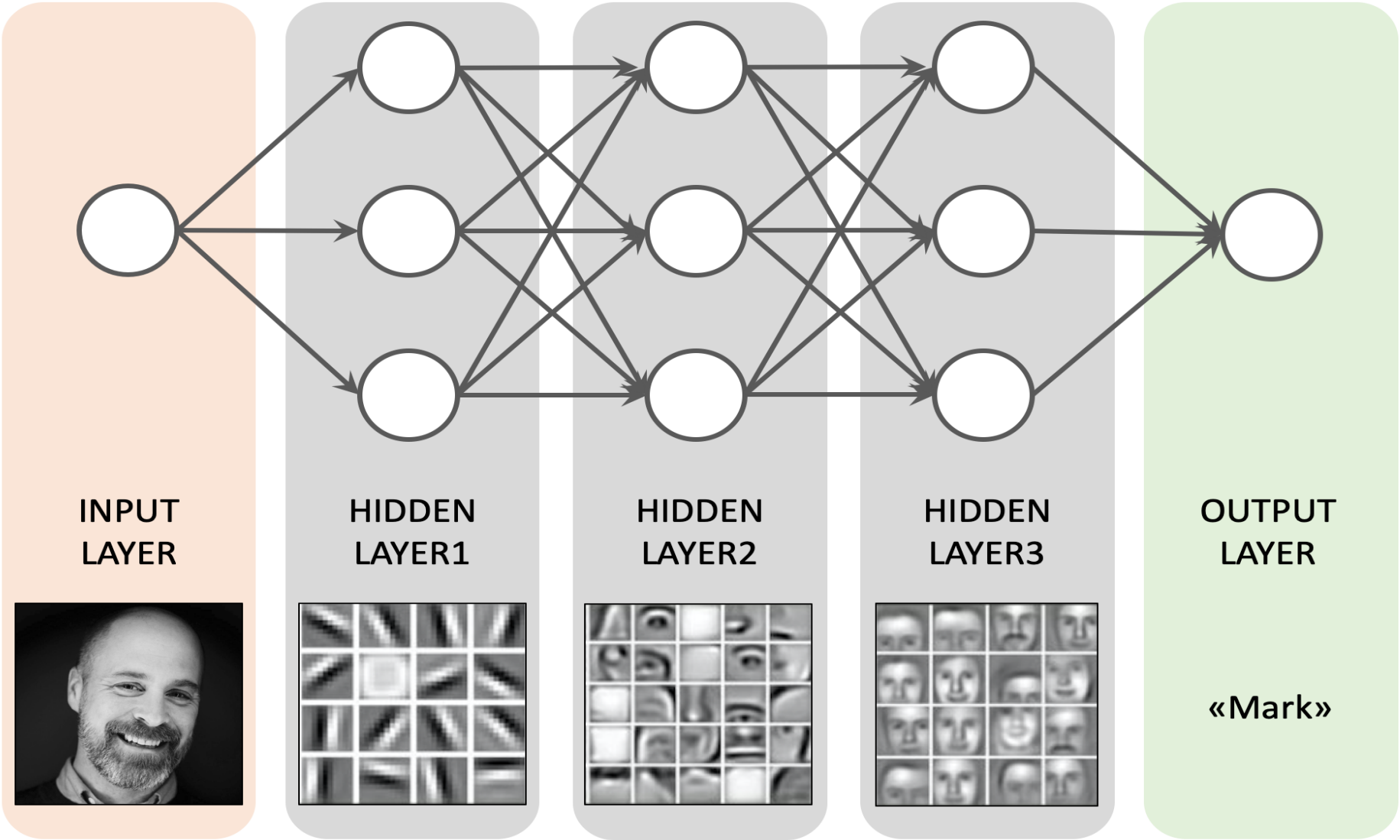
Идея

Разработать модель, которая научиться из исходных признаков извлекать новые и на их основе принимать итоговое решение

Персептрон с одним скрытым слоем



Персептрон



[0] https://www.bouvet.no/bouvet-deler/an-introduction-to-deep-learning

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

$y = f_o(\mathbf{w}^T \cdot \mathbf{h})$ – выходной слой. В качестве f_o обычно используется функция для принятия решения из линейных моделей:

- $y = \mathbf{w}^T \cdot \mathbf{h}$ – регрессия
- $y = \sigma(\mathbf{w}^T \cdot \mathbf{h}) = \dfrac{1}{1 + \exp -(\mathbf{w}^T \cdot \mathbf{h})}$ – бинарная классификация
- $\mathbf{y} = softmax(\mathbf{Wh}), \mathbf{W} \in \mathbb{R}^{c \times m}$ – классификация на $c > 2$ классов

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

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, которые мы хотим обучить
- Параметры θ должны быть такими, чтобы нейронная сеть выдавала правильные ответы на элементах обучающей выборки

Обучение нейронной сети с учителем

- Для оценки погрешности алгоритма выбирается функция ошибки $l = l(y_i, \tilde{y}_i), l \in \mathbb{R}$. Из раннее рассмотренных – MSE, cross-entropy, hinge
- 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 находить ее минимум по параметрам θ с помощью градиентного спуска: $\theta_{i+1} = \theta_i - \alpha \cdot \nabla L(\theta_i)$

SGD

- Вычисление L при больших n – ресурсоемкая задача
- Stochastic gradient descent (SGD) – на каждом шаге градиентного спуска случайно выбираем **один** пример 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, а следовательно процесс обучения может проходить долго
- Вычисление 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
- Потерю $l_j(\theta_i)$ можно считать параллельно по j

Обучение

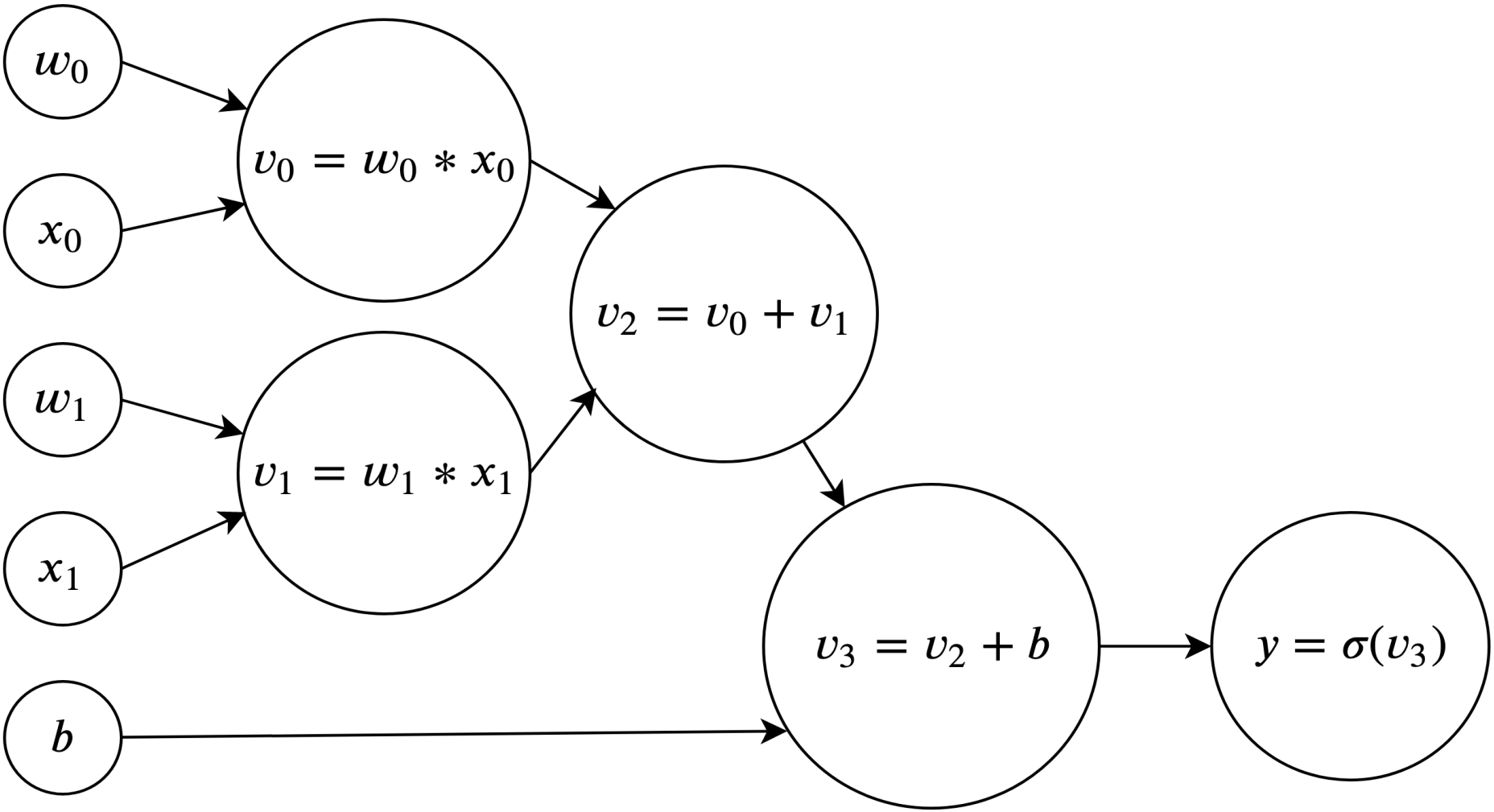
Обучение проводят некоторое количество “эпох” – полных итераций по обучающей выборке и останавливаются, когда качество на валидационной выборке перестает улучшаться
В настоящее время в основном используются адаптивные методы градиентного спуска, т.е. значение α на каждом шаге меняется в зависимости от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

Вопрос

Как вычислять градиент по каждому параметру сложной нейронной сети (в общем случае, сложной дифференцируемой функции)?

Граф вычислений

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



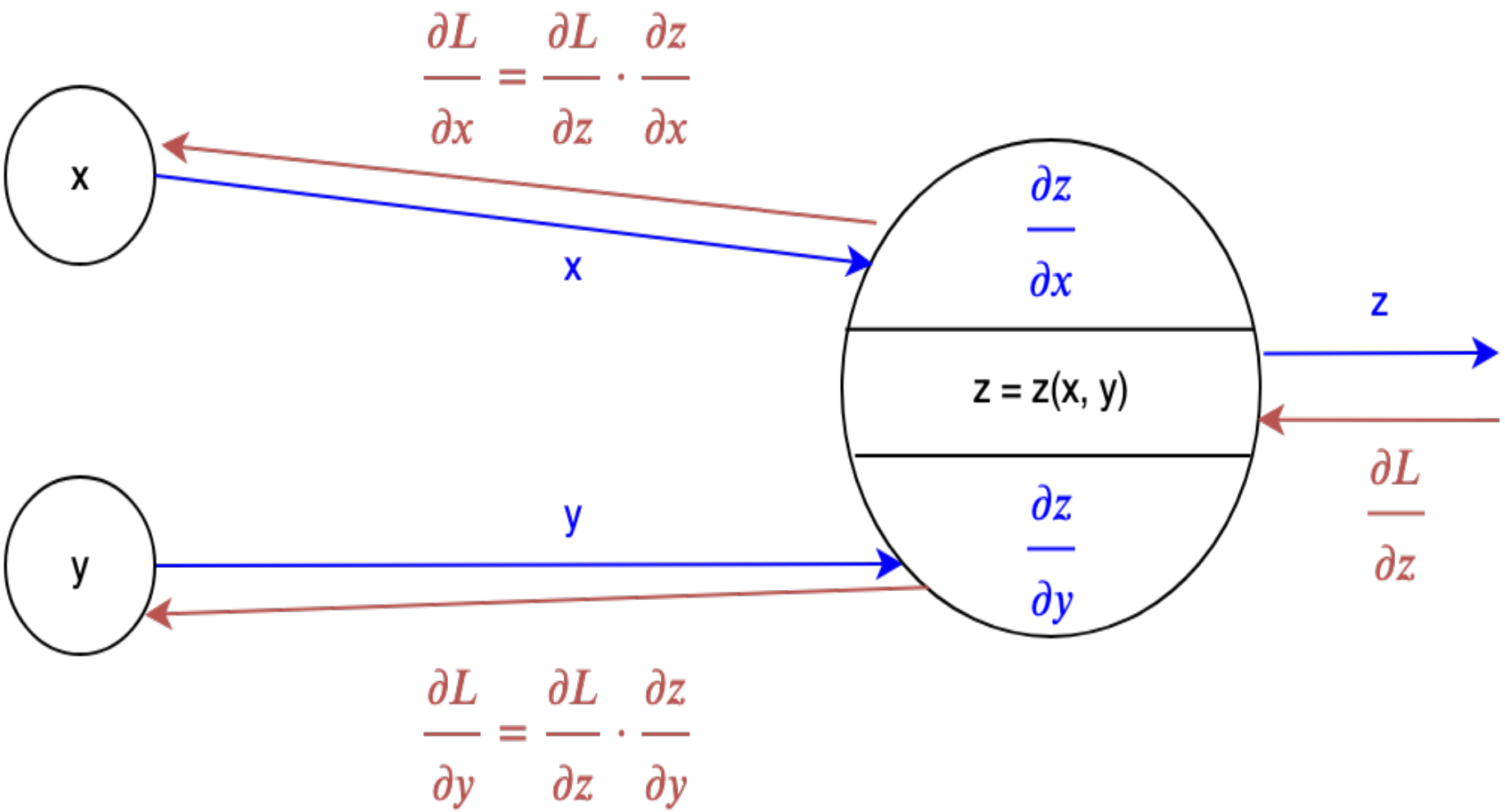
Backpropagation

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

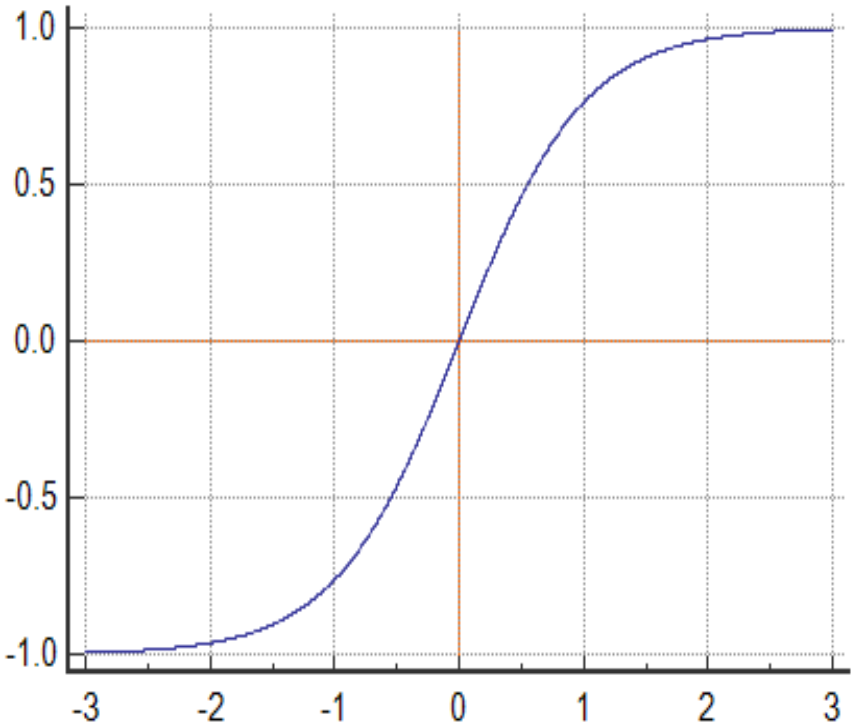


Проблема затухающих градиентов

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 нейронная сеть не сможет обучиться.

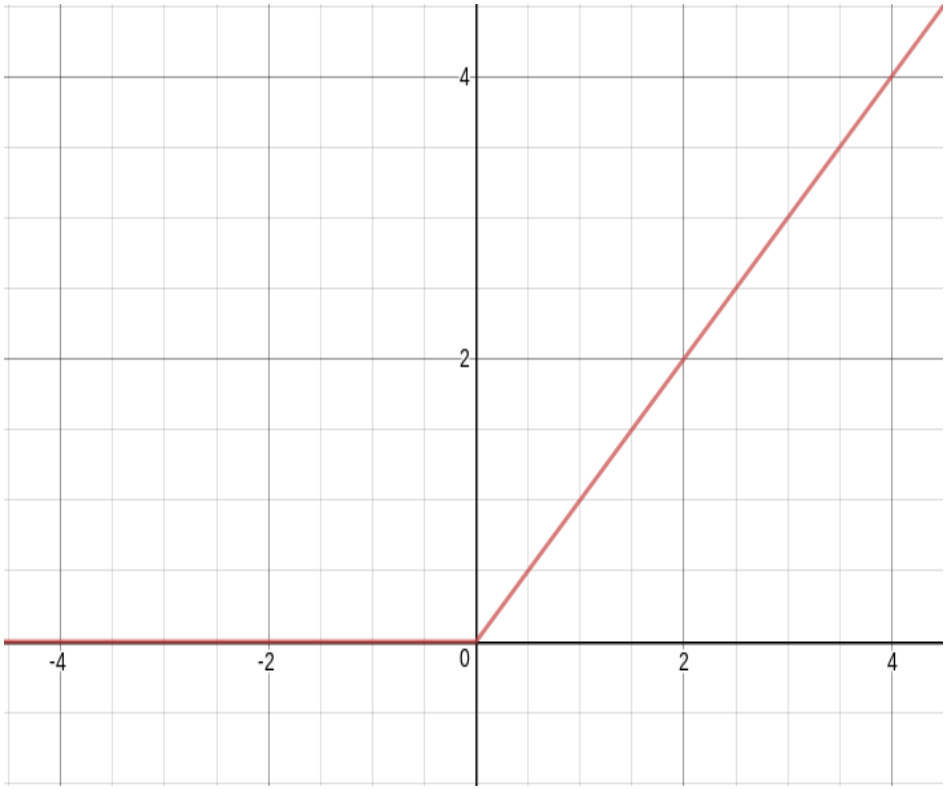
Чтобы с этим бороться, выбирают специальные функции активации и разрабатывают “хитрые” архитектуры.

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
- Широко используется в скрытых слоях

Начальная инициализация

Перед обучением нейронной сети нужно проинициализировать ее параметры θ. Обычно используется случайная инициализация небольшими 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 алгоритм $y : X \to \{0, 1\}$

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

TF-IDF признаки

- TF-IDF (TF — term frequency, IDF — inverse document frequency)
- tf-idf$(t, d, D) = tf(t, d) \cdot idf(t, D)$, где t – токен, d – текущий документ, D – обучающая выборка
- $tf(t, d) = \frac{n_t}{\sum_k n_k}$ – частота токена в данном документе
- $idf(t, D) = \log \frac{|D|}{|\{d_i \in D | t \in d_i\}|}$

TF-IDF признаки

- По обучающей выборке D строим словарь токенов V и считаем $idf(t, D)$ для каждого $t \in V$
- Теперь можем получить векторное представление $\mathbf{x}$ для произвольного текста d: $\mathbf{x} \in \mathbb{R}^{|V|}$, $x_i = \text{tf-idf}(t_i, d, D), i \in [1, |V|]$
- TF-IDF дает высокий вес токенам, часто встречающимся в конкретном документе, но обладающим низкой частотой относительно всей коллекции
- Таким образом, не обладающие смыслом слова (знаки препинания, соединительные союзы и т.д.) будут иметь низкий вес

Фреймворки для python

- sklearn.neural_network – недостаточная конфигурация
- tensorflow – высокий порог вхождения
- keras, pytorch

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))
```

Результаты

Усреднение по 5 запускам с разными случайными инициализациями весов:

- $accuracy = 0.955$
- $precision = 0.889$
- $recall = 0.766$
- $f1 = 0.822$

Статистика одного из запусков:

- $tp = 84$
- $tn = 717$
- $fp = 7$
- $fn = 28$
- $tp + fn = 112$ – положительные примеры
- $tn + fp = 724$ – отрицательные примеры

Предобработка

Исходные тексты содержат много незначимых цифр и большое число ссылок. Попробуем нормализовать текст: заменить все цифры на 0, все ссылки на <href>, чтобы конечное признаковое пространство не было разреженным и одинаковые по сути словоформы были действительно одинаковыми.

Результаты с предобработкой

Усреднение по 5 запускам с разными случайными инициализациями весов:

- $accuracy = 0.968 > 0.955$
- $precision = 0.905 > 0.889$
- $recall = 0.848 > 0.766$
- $f1 = 0.875 > 0.822$

Небольшая предобработка значительно улучшила качество метода

Полный pipeline

- https://github.com/vladosdudos/simple_spam_detection
- или https://nbviewer.jupyter.org/github/vladosdudos/simple_spam_detection/blob/master/simple_nns.ipynb

Разметка последовательности – NERC

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

IO-кодирование:

I-ЛИЧ I-ЛИЧ O O I-ЛОК

Чтобы подать каждое слово в нейронную сеть, нужно его представить вектором

One-hot кодирование

Пусть V – перенумерованное множество всех словоформ обучающей выборки (словарь).
Тогда каждому слову можно сопоставить вектор $\mathbf{x} \in \mathbb{R}^{|V|+1}$, где $x_i = 1$, если это i-е слово из словаря, $x_j = 0, j \neq i$.
Если слова нет в словаре (OOV – out of vocabulary), то для него зарезервирована отдельная позиция в данном векторе: $x_{|V|+1} = 1$

Кодирование с учетом контекста

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

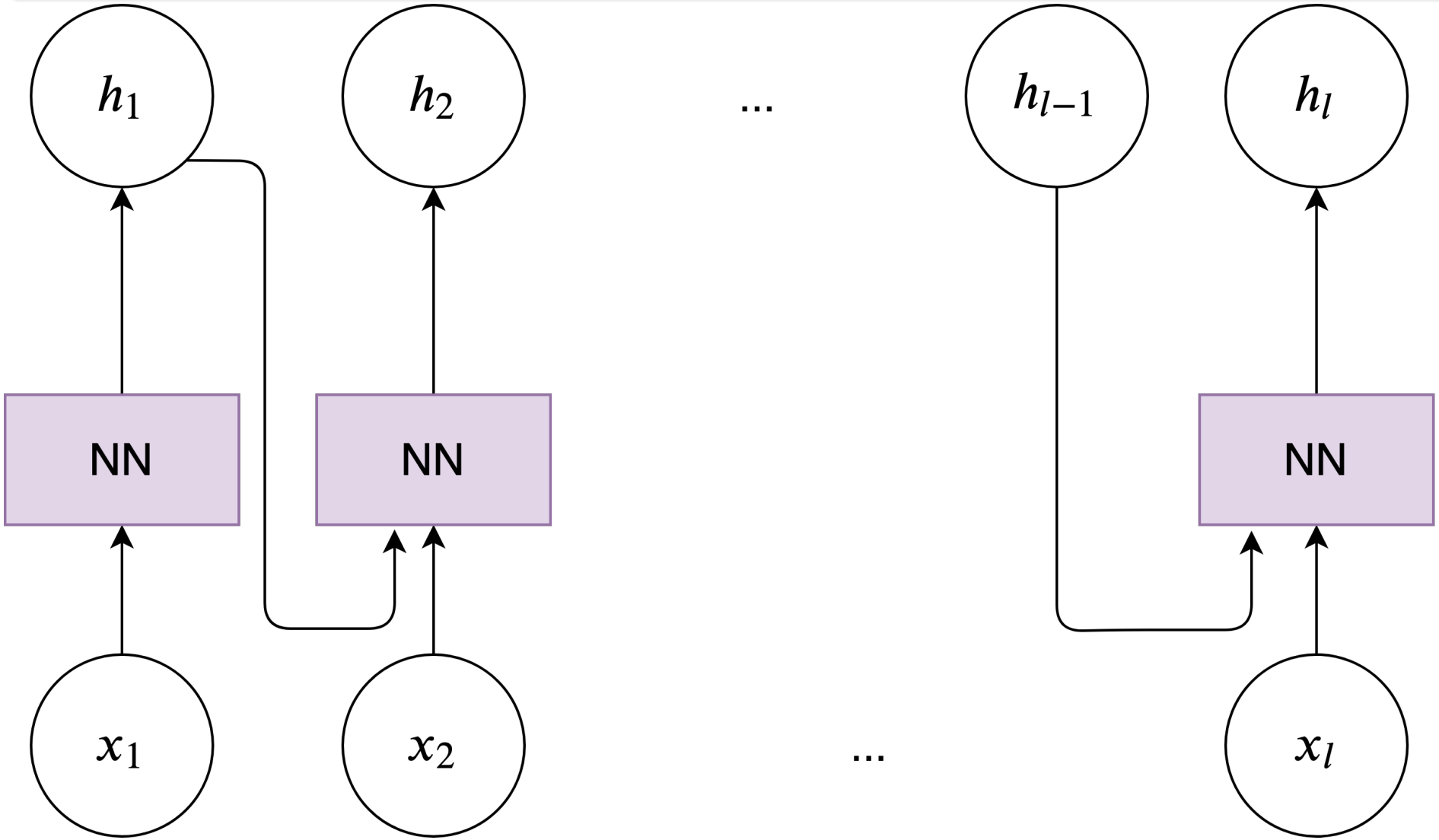
Как получить представление каждого слова, с учетом его нынешнего контекста, чтобы его классифицировать на предмет вхождения в сущность?

Кодирование с учетом контекста

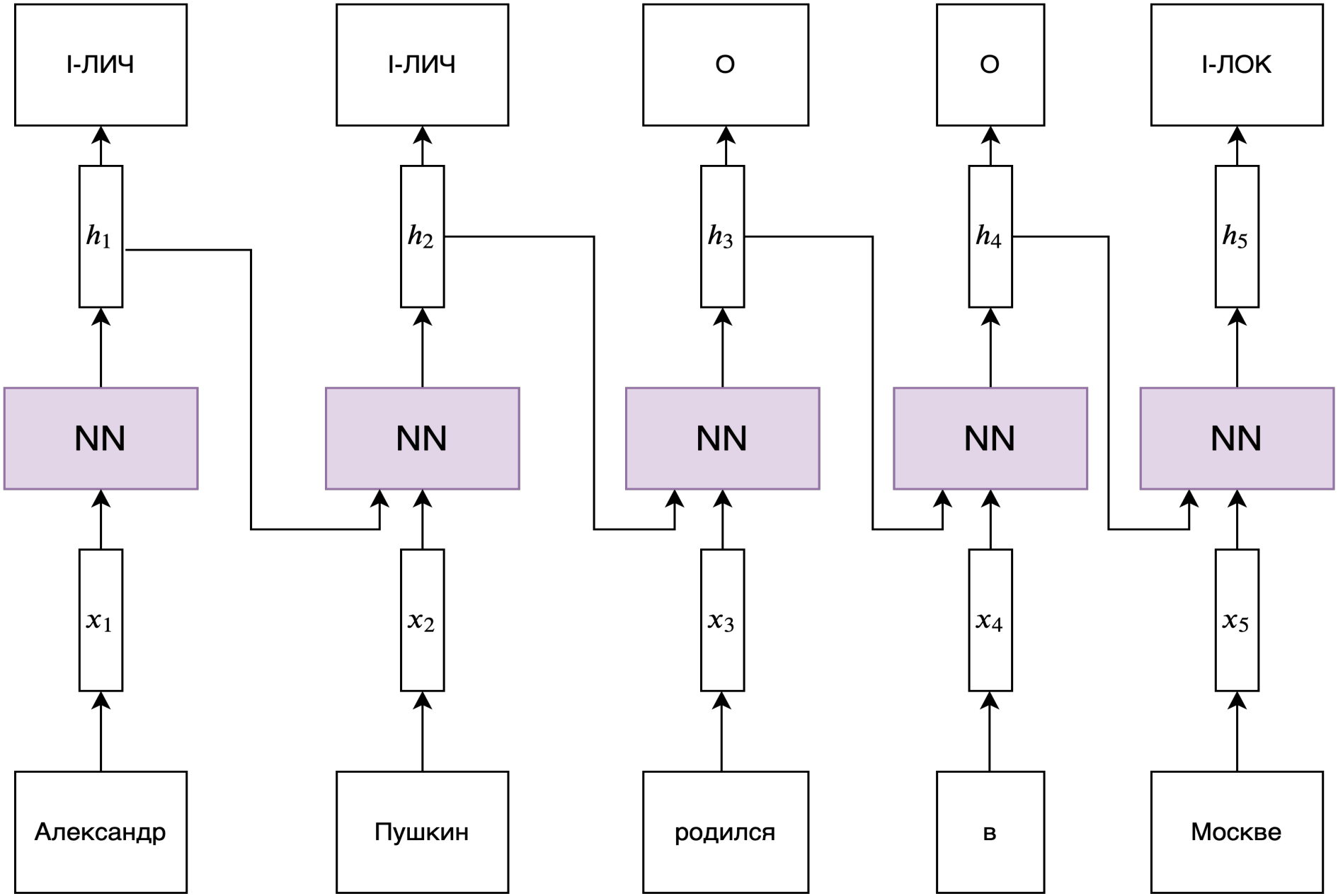
- сверточные нейронные сети (CNN)
- нейронные сети на основе механизма внимания (Transformer)
- **рекуррентные нейронные сети (GRU, LSTM)**

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 $\mathbf{b} = \mathbf{0}$:

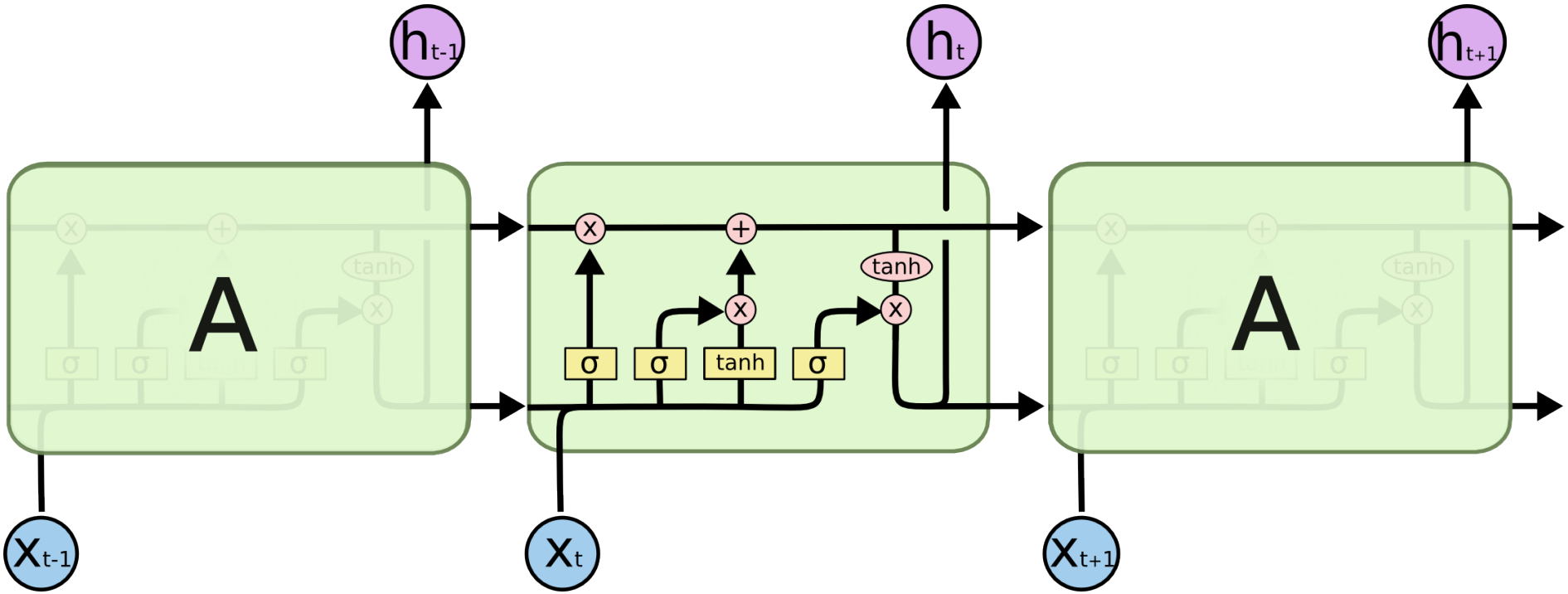
$$\mathbf{h_3} = f(\mathbf{W_x x_3} + \mathbf{W_h} f(\mathbf{W_x x_2} + \mathbf{W_h h_1}))$$

- Считая, что мы работаем со скалярами:

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

- Предыдущий выход не влияет на текущее решение нашей сети, а значит наша сеть забывает, что было раннее :(

LSTM



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

LSTM

- $\mathbf{f_t} = \sigma(\mathbf{W_f x_t} + \mathbf{U_f h_{t-1}} + \mathbf{b_f})$ – 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
  $\mathbf{\tilde{c}_t} = \tanh(\mathbf{W_c x_t} + \mathbf{U_c h_{t-1}} + \mathbf{b_c})$
  $\mathbf{c_t} = \mathbf{f_t} \odot \mathbf{c_{t-1}} + \mathbf{i_t} \odot \mathbf{\tilde{c}_t}$
  $\mathbf{h_t} = \mathbf{o_t} \odot \mathbf{\tanh}(\mathbf{c_t})$
- $\dfrac{\partial \mathbf{c_t}}{\partial \mathbf{c_{t-1}}} = diag(\mathbf{f_t})$ – 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 получить фиксированный вектор $\mathbf{a}$, 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, R.Socher CS224n Winter 2017 lectures
- Воронцов К.В. Машинное обучение

Вопросы?