Нейронные сети

Александр Сарачаков, Анастасия Кишкун

Сколтех, Москва, Россия

Skoltech
Skolkovo Institute of Science and Technology

Содержание

1. Мотивация
2. Персептрон
3. Нейронные сети
4. Обратное распространение ошибки

1 Мотивация

2 Персептрон

3 Нейронные сети

4 Обратное распространение ошибки

Нейроны I

- Функции мозга (мысли) возникают как результат включения **нейронов**
- Нейроны соединяются друг с другом с помощью **синапсов**, которые размножаются через **действие потенциала** (электрические импульсы), выпуская **нейротрансмиттеры**:
  - Синапсы могут быть **возбуждающие** (с увеличение потенциала) или **тормозящие** (с уменьшением потенциала), и имеют различные **уровни активации**
  - Обучение возникает как результат **пластичности** синапсов: они показывают долгосрочные изменения в прочности соединения
- В человеческом мозге существует около 10^{11} нейронов и около 10^{14} синапсов

Нейроны I

- Функции мозга (мысли) возникают как результат включения **нейронов**
- Нейроны соединяются друг с другом с помощью **синапсов**, которые размножаются через **действие потенциала** (электрические импульсы), выпуская **нейротрансмиттеры**:
  - Синапсы могут быть **возбуждающие** (с увеличение потенциала) или **тормозящие** (с уменьшением потенциала), и имеют различные **уровни активации**
  - Обучение возникает как результат **пластичности** синапсов: они показывают долгосрочные изменения в прочности соединения
- В человеческом мозге существует около 10^{11} нейронов и около 10^{14} синапсов

Нейроны I

- Функции мозга (мысли) возникают как результат включения **нейронов**
- Нейроны соединяются друг с другом с помощью **синапсов**, которые размножаются через **действие потенциала** (электрические импульсы), выпуская **нейротрансмиттеры**:
  - Синапсы могут быть **возбуждающие** (с увеличение потенциала) или **тормозящие** (с уменьшением потенциала), и имеют различные **уровни активации**
  - Обучение возникает как результат **пластичности** синапсов: они показывают долгосрочные изменения в прочности соединения
- В человеческом мозге существует около 10^{11} нейронов и около 10^{14} синапсов

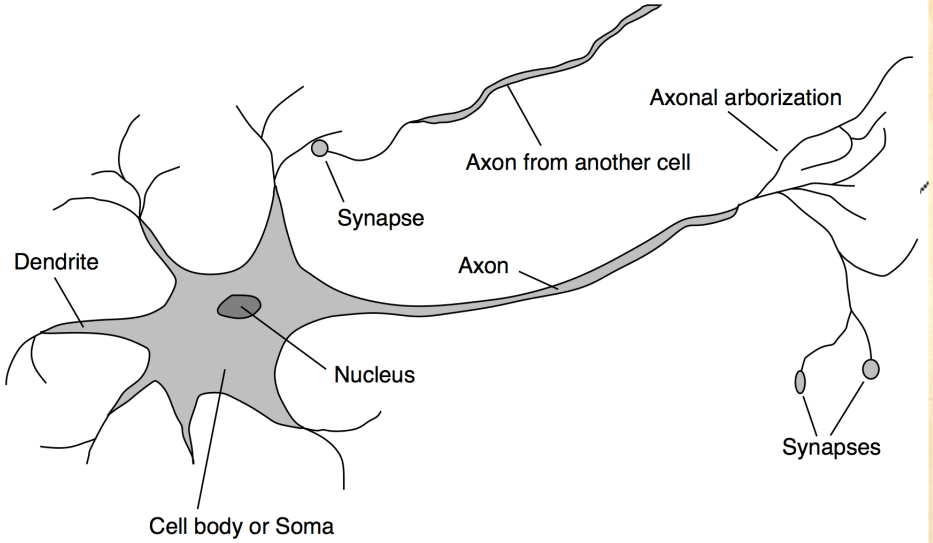
Нейроны I

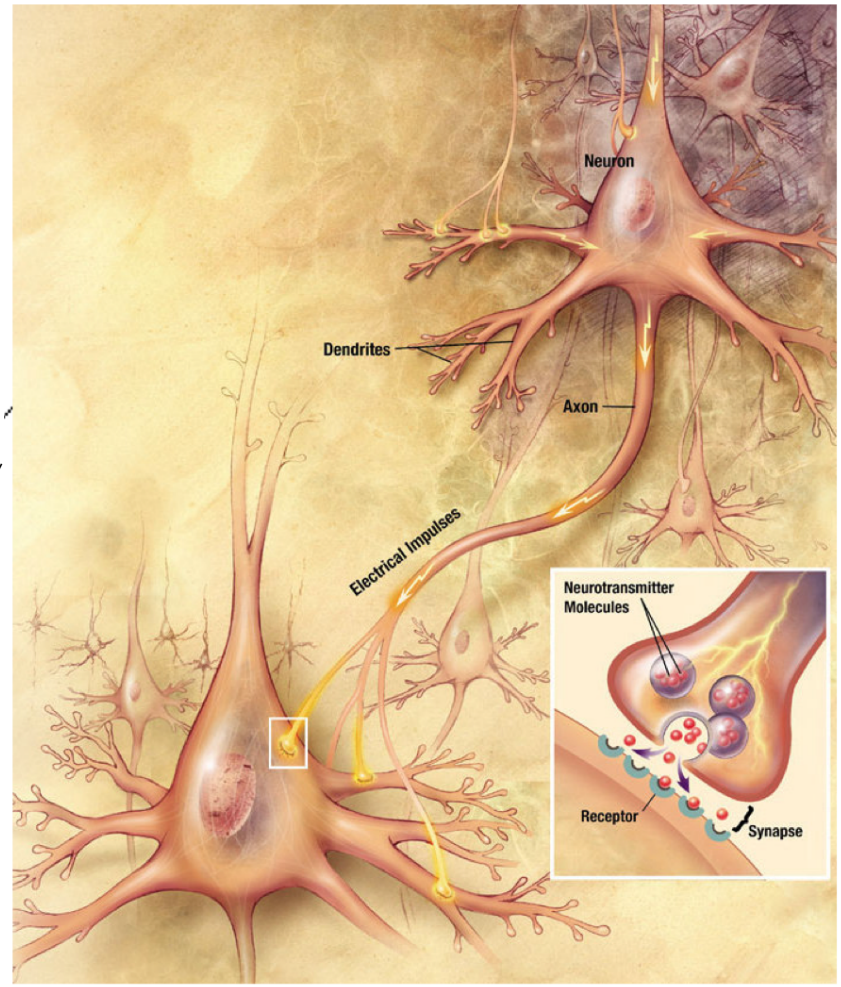
- Функции мозга (мысли) возникают как результат включения **нейронов**
- Нейроны соединяются друг с другом с помощью **синапсов**, которые размножаются через **действие потенциала** (электрические импульсы), выпуская **нейротрансмиттеры**:
  - Синапсы могут быть **возбуждающие** (с увеличение потенциала) или **тормозящие** (с уменьшением потенциала), и имеют различные **уровни активации**
  - Обучение возникает как результат **пластичности** синапсов: они показывают долгосрочные изменения в прочности соединения
- В человеческом мозге существует около 10^{11} нейронов и около 10^{14} синапсов

Нейроны I

- Функции мозга (мысли) возникают как результат включения **нейронов**
- Нейроны соединяются друг с другом с помощью **синапсов**, которые размножаются через **действие потенциала** (электрические импульсы), выпуская **нейротрансмиттеры**:
  - Синапсы могут быть **возбуждающие** (с увеличение потенциала) или **тормозящие** (с уменьшением потенциала), и имеют различные **уровни активации**
  - Обучение возникает как результат **пластичности** синапсов: они показывают долгосрочные изменения в прочности соединения
- В человеческом мозге существует около 10^{11} нейронов и около 10^{14} синапсов

Нейроны II


Axonal arborization
Axon from another cell
Synapse
Dendrite
Axon
Nucleus
Synapses
Cell body or Soma


Neuron
Dendrites
Axon
Electrical impulses
Neurotransmitter Molecules
Receptor
Synapse

Структура мозга

- Различные области мозга имеют различные функции
  - Некоторые области имеют одну и ту же функцию у всех людей (например, область Брока для моторной речи); общая компоновка обычно целостная
  - Некоторые области более пластичны, и изменчивы в своих функциях; также, низкоуровневые структура и функции сильно варьируются
- Мы не знаем, насколько различны функции, которые присваиваются или приобретаются
  - Частично результат физической компоновки/подключения к входам (датчикам) и выходам (эффекторам)
  - Частично результат опыта (обучения)
- Мы не понимаем, как эта нейронная структура приводит к тому, что мы воспринимаем как "сознание"или"мысль"

Структура мозга

- Различные области мозга имеют различные функции
  - Некоторые области имеют одну и ту же функцию у всех людей (например, область Брока для моторной речи); общая компоновка обычно целостная
  - Некоторые области более пластичны, и изменчивы в своих функциях; также, низкоуровневые структура и функции сильно варьируются
- Мы не знаем, насколько различны функции, которые присваиваются или приобретаются
  - Частично результат физической компоновки/подключения к входам (датчикам) и выходам (эффекторам)
  - Частично результат опыта (обучения)
- Мы не понимаем, как эта нейронная структура приводит к тому, что мы воспринимаем как "сознание"или"мысль"

Структура мозга

- Различные области мозга имеют различные функции
  - Некоторые области имеют одну и ту же функцию у всех людей (например, область Брока для моторной речи); общая компоновка обычно целостная
  - Некоторые области более пластичны, и изменчивы в своих функциях; также, низкоуровневые структура и функции сильно варьируются
- Мы не знаем, насколько различны функции, которые присваиваются или приобретаются
  - Частично результат физической компоновки/подключения к входам (датчикам) и выходам (эффекторам)
  - Частично результат опыта (обучения)
- Мы не понимаем, как эта нейронная структура приводит к тому, что мы воспринимаем как "сознание"или"мысль"

Гипотеза “одного алгоритма обучения”

- Есть некоторые доказательства того, что человеческий мозг использует по существу один и тот же алгоритм, чтобы понять множество различных способов ввода
- Пример: эксперименты над хорьками, в которых "вход"для зрения был подключен к слуховой части мозга, а слуховая кора училась "видеть"

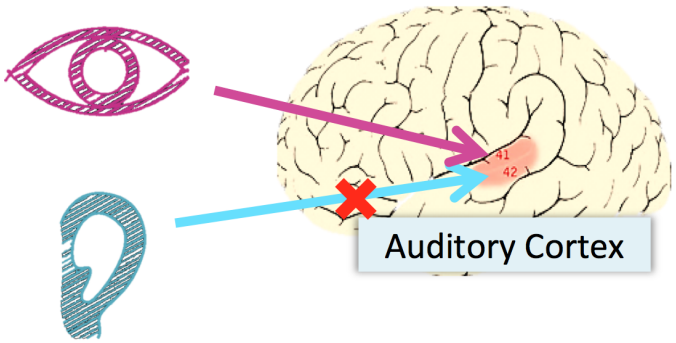


Рис. – Слуховая кора головного мозга учится видеть

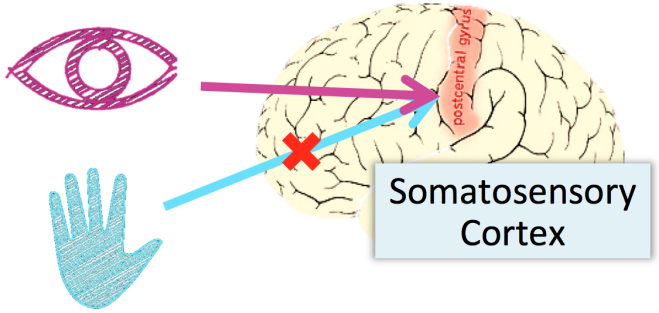


Рис. – Соматосенсорная кора головного мозга учится видеть

Гипотеза “одного алгоритма обучения”

- Есть некоторые доказательства того, что человеческий мозг использует по существу один и тот же алгоритм, чтобы понять множество различных способов ввода
- Пример: эксперименты над хорьками, в которых "вход"для зрения был подключен к слуховой части мозга, а слуховая кора училась "видеть"

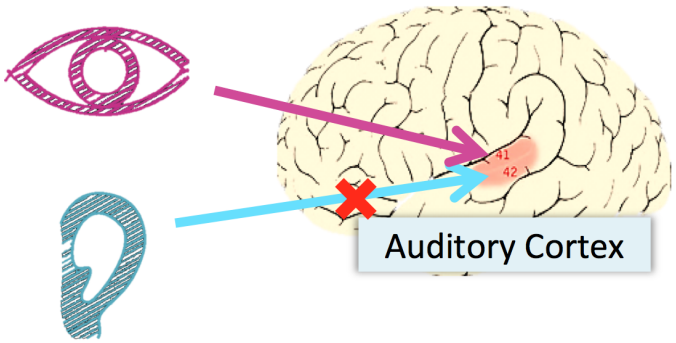


Рис. – Слуховая кора головного мозга учится видеть

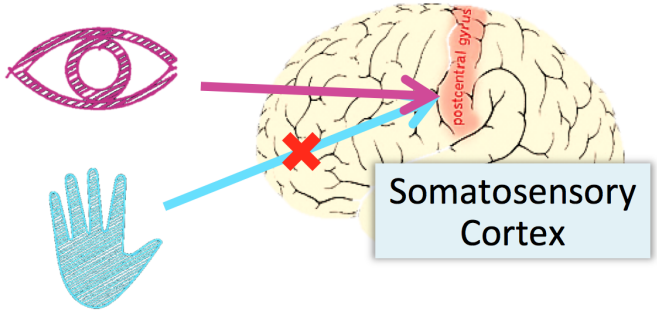


Рис. – Соматосенсорная кора головного мозга учится видеть

Представление сенсоров в мозге

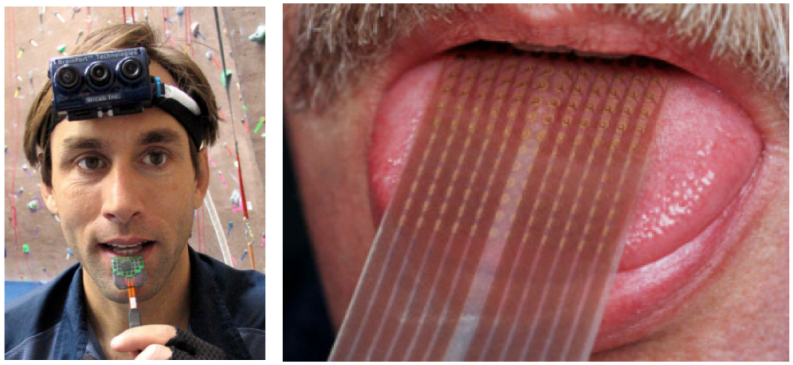


Рис. – "Зрение"с помощью языка

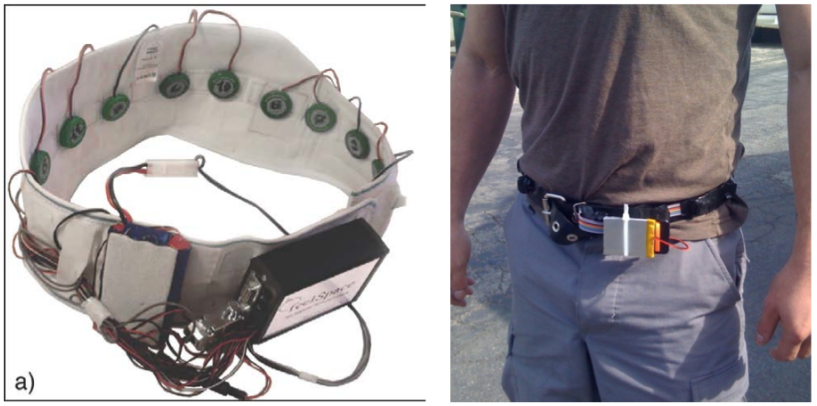


Рис. – Тактильный пояс: ощущение направления

Сравнение вычислительных мощностей

- Компьютеры гораздо быстрее нейронов, т.е.: время цикла 10^{-9} сек для компьютеров против 10^{-3} сек для человеческого мозга
- Но в человеческом мозгу гораздо больше нейронов(10^{11} нейронов, 10^{14} синапсов), чем мы можем смоделировать на компьютере, и все они работают параллельно (пропускная способность 10^{9} бит/сек для компьютеров против 10^{14} бит/сек для человеческого мозга)
- Нейронные сети рассчитаны на массовую параллельность
- Мозг в миллиард раз быстрее

Сравнение вычислительных мощностей

- Компьютеры гораздо быстрее нейронов, т.е.: время цикла 10^{-9} сек для компьютеров против 10^{-3} сек для человеческого мозга
- Но в человеческом мозгу гораздо больше нейронов(10^{11} нейронов, 10^{14} синапсов), чем мы можем смоделировать на компьютере, и все они работают параллельно (пропускная способность 10^{9} бит/сек для компьютеров против 10^{14} бит/сек для человеческого мозга)
- Нейронные сети рассчитаны на массовую параллельность
- Мозг в миллиард раз быстрее

Сравнение вычислительных мощностей

- Компьютеры гораздо быстрее нейронов, т.е.: время цикла 10^{-9} сек для компьютеров против 10^{-3} сек для человеческого мозга
- Но в человеческом мозгу гораздо больше нейронов(10^{11} нейронов, 10^{14} синапсов), чем мы можем смоделировать на компьютере, и все они работают параллельно (пропускная способность 10^{9} бит/сек для компьютеров против 10^{14} бит/сек для человеческого мозга)
- Нейронные сети рассчитаны на массовую параллельность
- Мозг в миллиард раз быстрее

Нейронные сети

- Происхождение: алгоритмы, которые пытаются имитировать мозг
- Широко использовались в 80-х и начале 90-х; популярность уменьшилась в конце 90-х
- Недавнее возрождение: современный методы для многих приложений
- Искусственные нейронные сети не так сложны или запутаны, как реальная структура мозга

1 Мотивация

2 Персептрон

3 Нейронные сети

4 Обратное распространение ошибки

Розенблатт и его персептрон I

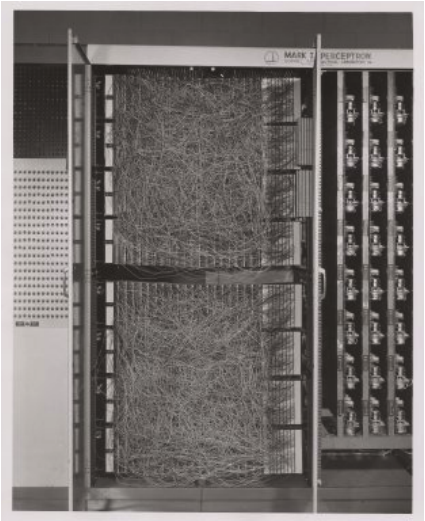
Лаборатория Cornell Aeronautical

Рис. – Розенблатт & Марк И. персептрон: первая машина, которая может “выучить” распознавание и идентификацию цифровых паттернов

- Изобретён Ф. Розенблаттом в 1957 в попытках понять человеческую память, обучение и когнитивные процессы
- Первая вычислительная нейронная сеть с алгоритмом обучения
- Стала основой исследования распознавания образов

Одна из самых ранних и влиятельных нейронных сетей: важная веха в ИИ

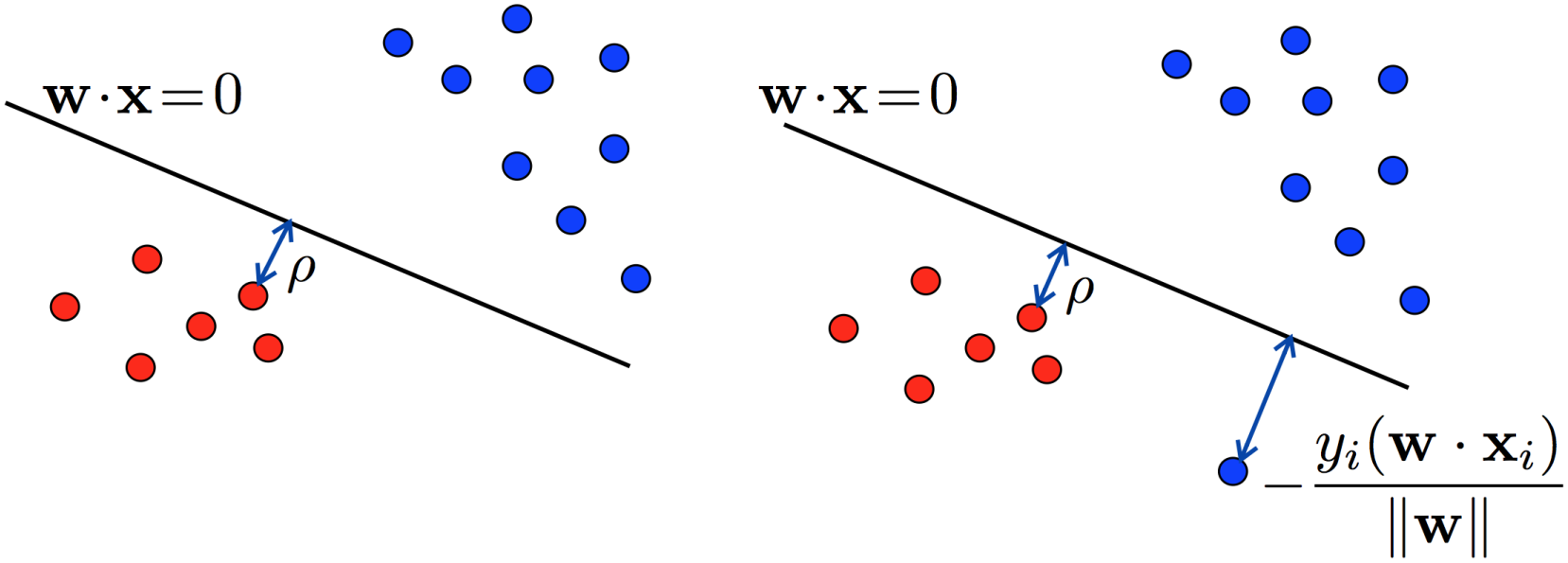
Розенблатт и его персептрон II



- Машина The Mark I Perceptron была присоединена к камере с 20×20 фотоэлементами сульфида кадмия для конструирования изображений из 400 пикселей. Основной видимой особенностью является плата, которая позволяет экспериментировать с различными комбинациями входных объектов. Справа от этого находятся массивы потенциометров, реализующих адаптивные веса
- The New York Times написали, что это был “зародыш электронного компьютера, который в будущем сможет ходить, говорить, видеть, разговаривать, писать, чинить себя и знать о своём существовании”. Проще говоря, искусственный интеллект

Разделяющая гиперплоскость

- Отступ (Margin) и ошибки



Алгоритм Персептрон

Perceptron($\mathbf{w}_0$)

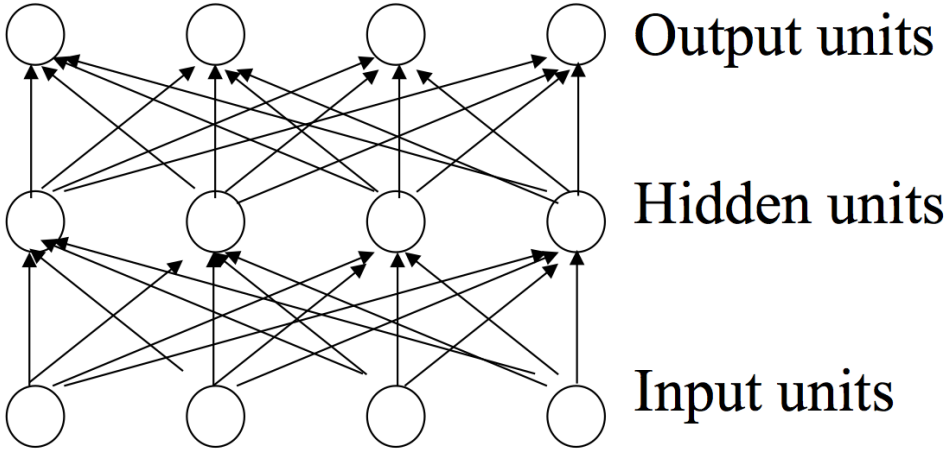
1. $\mathbf{w}_1 \leftarrow \mathbf{w}_0$ (обычно $\mathbf{w}_0 = 0$)
2. **for** $t \leftarrow 1$ **to** T **do**
3. Receive($\mathbf{x}_t$)
4. $\widehat{y}_t \leftarrow \operatorname{sgn}(\mathbf{w}_t \cdot \mathbf{x}_t)$
5. Receive(y_t)
6. **if** $(\widehat{y}_t \neq y_t)$ **then**
7. $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_t \mathbf{x}_t$ (or $\leftarrow \mathbf{w}_t + \eta y_t \mathbf{x}_t$, $\eta > 0$)
8. **else** $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t$
9. **return** $\mathbf{w}_{T+1}$

1 Мотивация

2 Персептрон

3 Нейронные сети

4 Обратное распространение ошибки

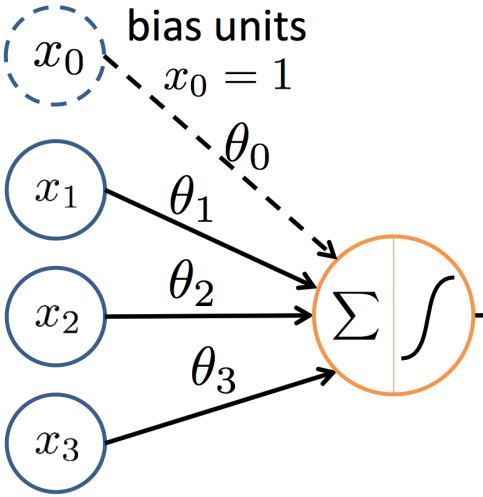


Layered feed-forward network

- Нейронные сети состоят из **вершин** или **юнитов**, связанных **ссылками**
- Каждая ссылка имеет соответствующий **вес** и **уровень активации**
- У каждой вершины есть **входная функция** (обычно суммирование взвешенного входа), **функция активации**, и **выход**

Нейронные сети: Логистический (сигмоидный) юнит

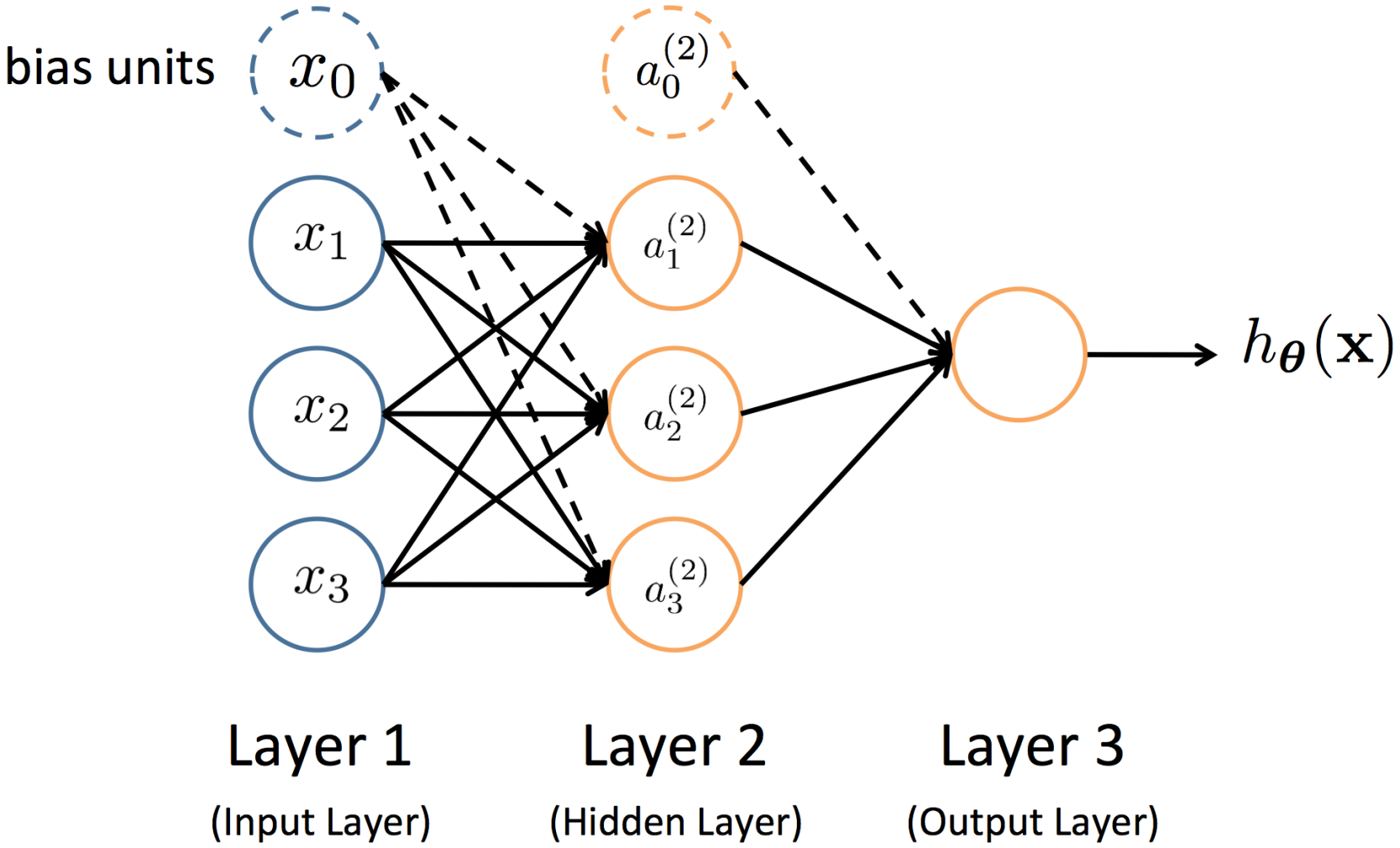
- Обозначим $\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$ и $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$



$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}\right) = \frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}}}$$

- Сигмоидная (логистическая) функция активации $g(z) = \frac{1}{1+e^{-z}}$

Многослойная нейросеть



Процесс прямого распространения

- Юниты входного слоя задаются некоторой внешней функцией, которая приводит к тому, что их выходные связи будут **активированы** на определённом уровне
- Проходя вперёд по сети, **входная функция** каждого юнита применяется для вычисления входного значения
  - Обычно это просто взвешенная сумма активаций на связях, попадающих в эту вершину
- **Функция активации** трансформирует эту входную функцию в финальное значение
  - Обычно это **нелинейная** функция, часто **сигмоида**, соответствующая “порогу” данной вершины

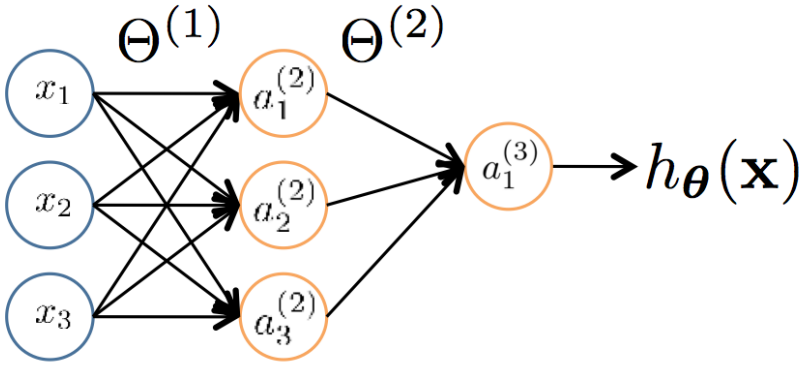
Процесс прямого распространения

- Юниты входного слоя задаются некоторой внешней функцией, которая приводит к тому, что их выходные связи будут **активированы** на определённом уровне
- Проходя вперёд по сети, **входная функция** каждого юнита применяется для вычисления входного значения
  - Обычно это просто взвешенная сумма активаций на связях, попадающих в эту вершину
- **Функция активации** трансформирует эту входную функцию в финальное значение
  - Обычно это **нелинейная** функция, часто **сигмоида**, соответствующая "порогу" данной вершины

Процесс прямого распространения

- Юниты входного слоя задаются некоторой внешней функцией, которая приводит к тому, что их выходные связи будут **активированы** на определённом уровне
- Проходя вперёд по сети, **входная функция** каждого юнита применяется для вычисления входного значения
  - Обычно это просто взвешенная сумма активаций на связях, попадающих в эту вершину
- **Функция активации** трансформирует эту входную функцию в финальное значение
  - Обычно это **нелинейная** функция, часто **сигмоида**, соответствующая “порогу” данной вершины

Нейросеть: уравнения



- $a_i^{(j)}$ = "активации" юнита i в слое j
- $\Theta^{(j)}$ = матрица весов, контролирующая соответствие слоя j слою $j+1$

$$a_1^{(2)} = g\left(\theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3\right)$$
$$a_2^{(2)} = g\left(\theta_{20}^{(1)}x_0 + \theta_{21}^{(1)}x_1 + \theta_{22}^{(1)}x_2 + \theta_{23}^{(1)}x_3\right)$$
$$a_3^{(2)} = g\left(\theta_{30}^{(1)}x_0 + \theta_{31}^{(1)}x_1 + \theta_{32}^{(1)}x_2 + \theta_{33}^{(1)}x_3\right)$$
$$h_\theta(\mathbf{x}) = a_1^{(3)} = g\left(\theta_{10}^{(2)}a_0^{(2)} + \theta_{11}^{(2)}a_1^{(2)} + \theta_{12}^{(2)}a_2^{(2)} + \theta_{13}^{(2)}a_3^{(2)}\right)$$

Если у нейросети s_j юнитов в слое j и s_{j+1} юнитов в слое $j+1$, тогда $\Theta^{(j)}$ имеет размерность $s_{j+1} \times (s_j + 1)$: $\Theta^{(1)} \in \mathbb{R}^{3\times 4}$, $\Theta^{(2)} \in \mathbb{R}^{1\times 4}$

Pytorch: имплементация

```python
# зададим класс для линейной регрессии. Мы наследуем его от torch.nn.Module
# нам нужны две функции:
# * __init__ определяет, как мы создаем объект этого класса
# * forward определяет, как мы делаем прогноз

class linearRegression(torch.nn.Module):
  # инициализация объекта класса
    def __init__(self, input_dimension, output_dimension):
        # сначала запустим init аналогичный родительскому классу
        super(linearRegression, self).__init__()
        # затем создадим свой первый слой на pytorch
        # так как модель у нас линейная, то и слой линейный
        self.linear1 = torch.nn.Linear(input_dimension, 5)
        self.linear2 = torch.nn.Linear(5, output_dimension)

  # подсчет прогноза
    def forward(self, x):
        out = self.linear1(x)
        return self.linear2(out)

    def predict(self, test):
        return self.__call__(test)
```

```python
output_dimension = 1

model = linearRegression(5, output_dimension)

# используем GPU, чтобы все считалось быстрее
if torch.cuda.is_available():
    model.cuda()
```

```python
model.linear1.weight, model.linear1.bias
```

executed in 51ms, finished 18:23:31 2021-04-06

```
(Parameter containing:
 tensor([[-0.4000,  0.4447, -0.2409,  0.0925,  0.0708],
         [-0.2905, -0.1201,  0.3794, -0.3067, -0.1369],
         [-0.2290,  0.2747,  0.4061,  0.0181, -0.1600],
         [-0.2440, -0.3070, -0.3737, -0.3610, -0.4309],
         [-0.0929, -0.2712,  0.0623, -0.3621, -0.3247]], device='cuda:0',
        requires_grad=True),
 Parameter containing:
 tensor([-0.3787, -0.0727, -0.4465,  0.0838,  0.4177], device='cuda:0',
        requires_grad=True))
```

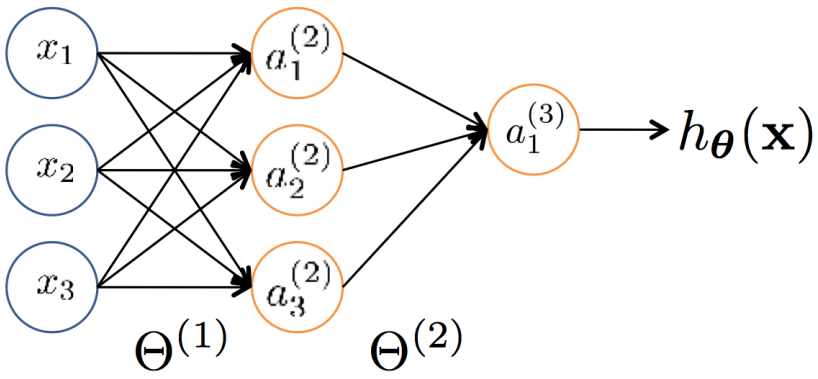
Нейросети: векторный формат

$$a_1^{(2)} = g\left(\theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3\right) = g(z_1^{(2)})$$

$$a_2^{(2)} = g\left(\theta_{20}^{(1)}x_0 + \theta_{21}^{(1)}x_1 + \theta_{22}^{(1)}x_2 + \theta_{23}^{(1)}x_3\right) = g(z_2^{2})$$

$$a_3^{(2)} = g\left(\theta_{30}^{(1)}x_0 + \theta_{31}^{(1)}x_1 + \theta_{32}^{(1)}x_2 + \theta_{33}^{(1)}x_3\right) = g(z_3^{(2)})$$

$$h_\theta(\mathbf{x}) = a_1^{(3)} = g\left(\theta_{10}^{(2)}a_0^{(2)} + \theta_{11}^{(2)}a_1^{(2)} + \theta_{12}^{(2)}a_2^{(2)} + \theta_{13}^{(2)}a_3^{(2)}\right) = g(z_1^{(3)})$$



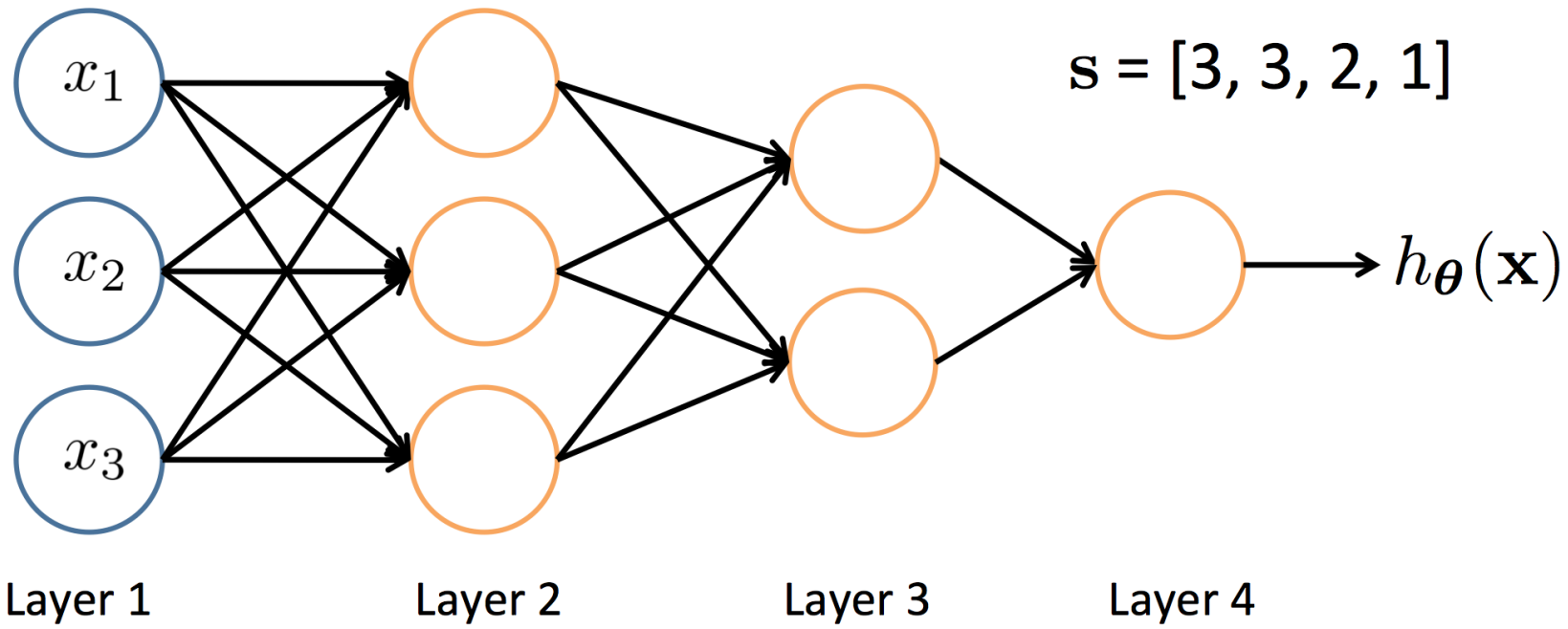
Шаги прямого распространения:

- $\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{x}$
- $\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$

Добавляем $a_0^{(2)} = 1$

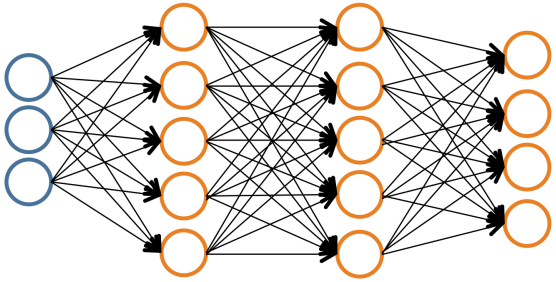
- $\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$
- $h_\theta(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^3)$

Другие архитектуры нейронных сетей



- L обозначает количество слоёв
- $\mathbf{s} \in (\mathbb{N}_+)^L$ содержит количество вершин в каждом слое
  - Не считаем bias юниты
  - Обычно, $s_0 = N$ (количество входных признаков) и $s_{L-1} = K$ (количество классов)

Несколько выходов: один-против-всех I



$$h_\theta(\mathbf{x}) \in \mathbb{R}^K$$

$$h_\theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

когда кошка

$$h_\theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

когда собака

$$h_\theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

когда машина

$$h_\theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

когда водитель

Несколько выходов: один-против-всех II

Рис. – Кошка

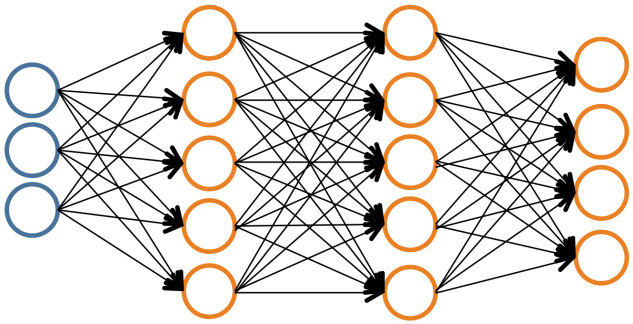
Рис. – Собака

Рис. – Машина

Рис. – Водитель

- Имеем $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$
- Конвертируем метки в представление 1-из-K, т.е. $y_i = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ когда машина, и т.д.

Классификация нейросетью



Имеем

- $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$
- $\mathbf{s} \in (\mathbb{N}_+)^L$ содержит количество нейронов (юнитов) на каждом слое, $s_0 = N$ (количество признаков)

Бинарная классификация:

— $y = 0$ или 1

— 1 выходной нейрон ($s_{L-1} = 1$)

Многоклассовая классификация (K классов)

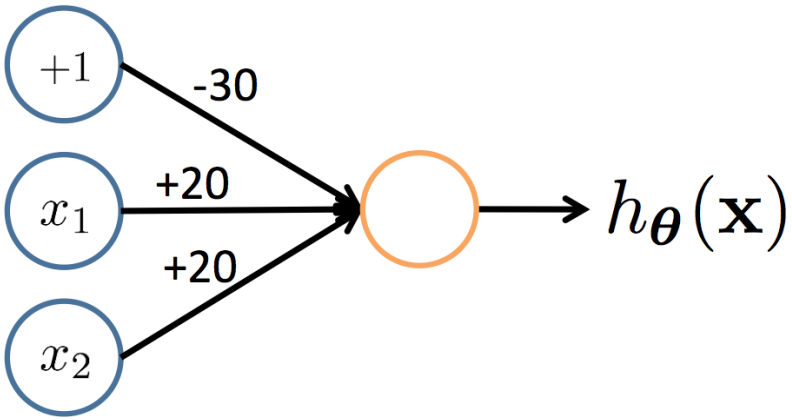
— $y \in \mathbb{R}^K$, т.е. $y = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ когда машина

— K выходных нейронов ($s_{L-1} = K$)

Simple example: AND

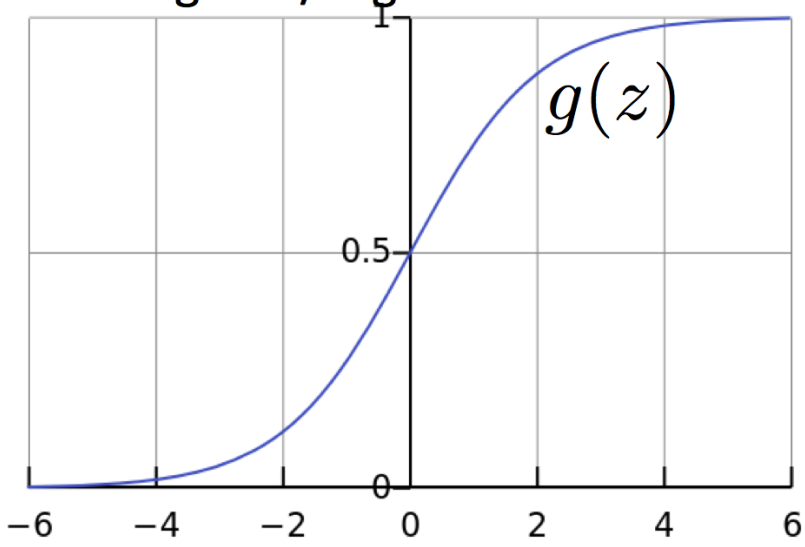
$x_1, x_2 \in \{0, 1\}$

$y = x_1 \text{ AND } x_2$



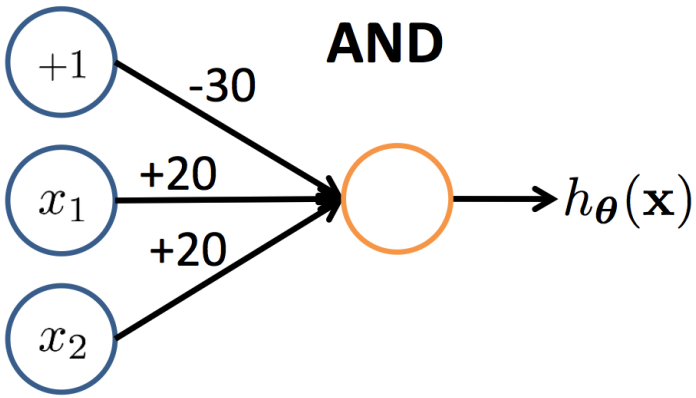
$$\mathrm{h}_\Theta(\mathbf{x}) = g(\text{-}30 + 20x_1 + 20x_2)$$

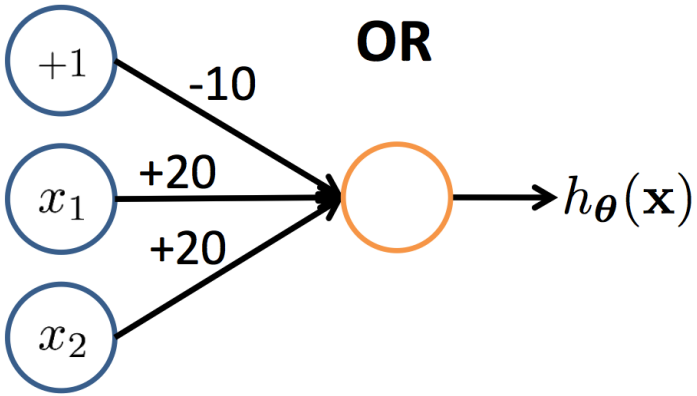
Logistic / Sigmoid Function

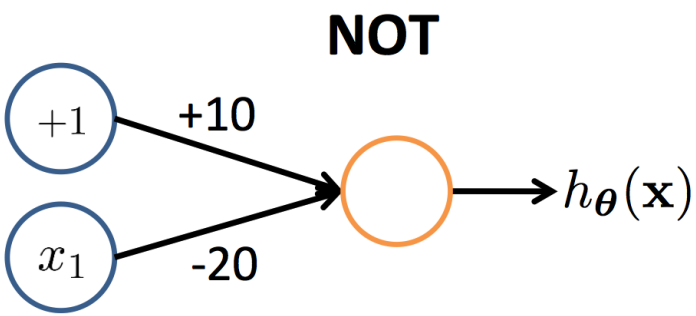


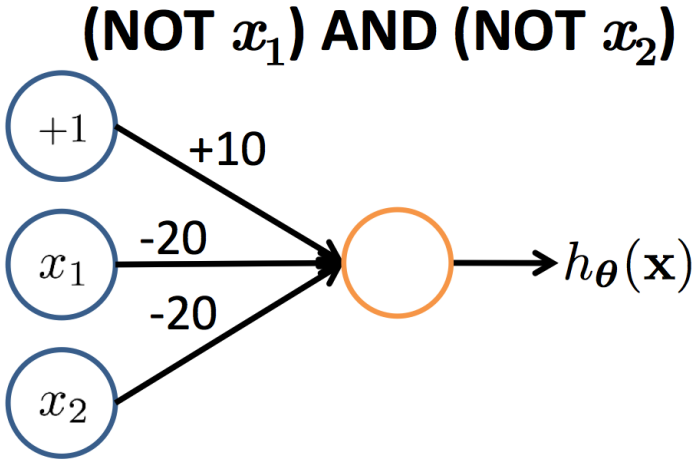
x_1	x_2	$\mathrm{h}_\Theta(\mathbf{x})$
0	0	$g(\text{-}30) \approx 0$
0	1	$g(\text{-}10) \approx 0$
1	0	$g(\text{-}10) \approx 0$
1	1	$g(10) \approx 1$

Представление булевых функций II



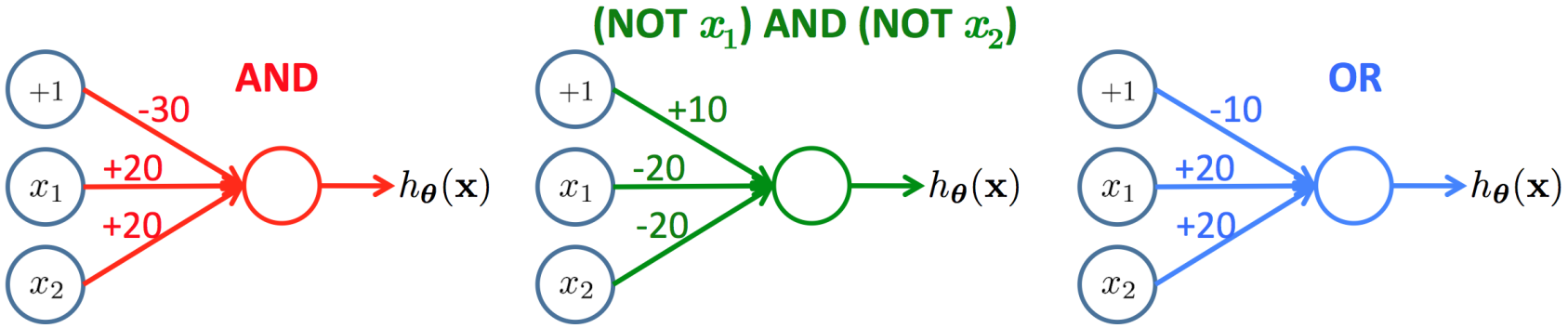




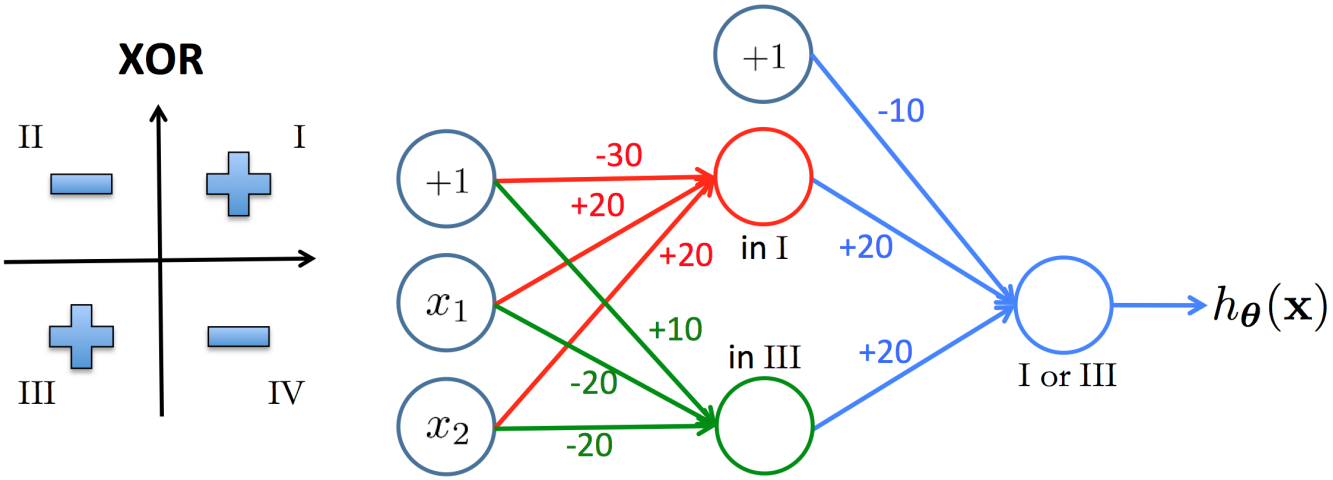


Комбинирование представлений для нелинейности

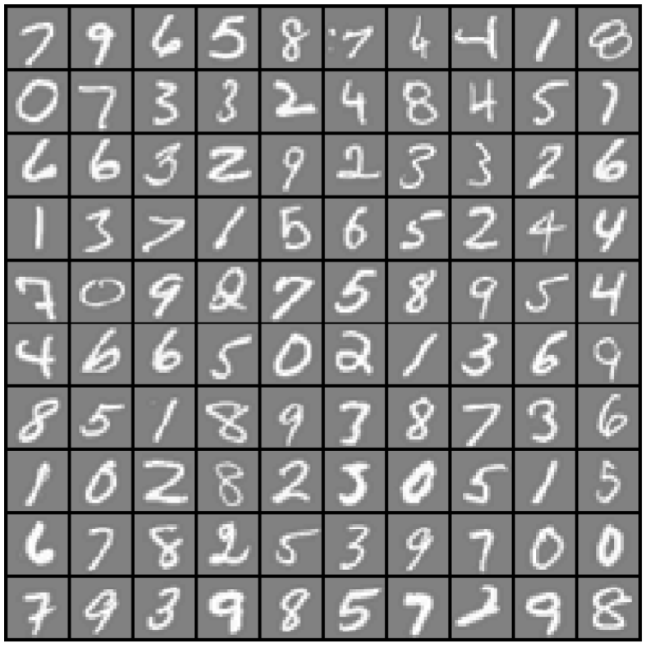
Строительные блоки:



XOR функция:

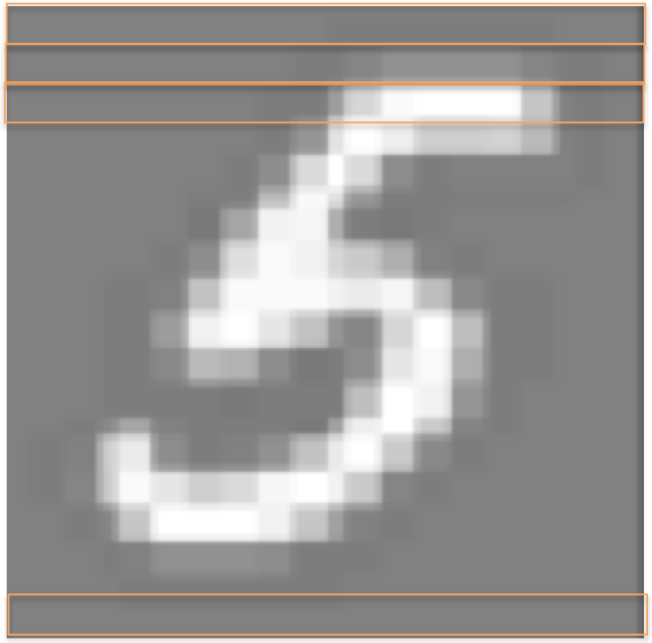


Представление слоёв I



20 × 20 pixel images

$d = 400$ 10 classes



$x_1 \ \dots \ x_{20}$

$x_{21} \ \dots \ x_{40}$

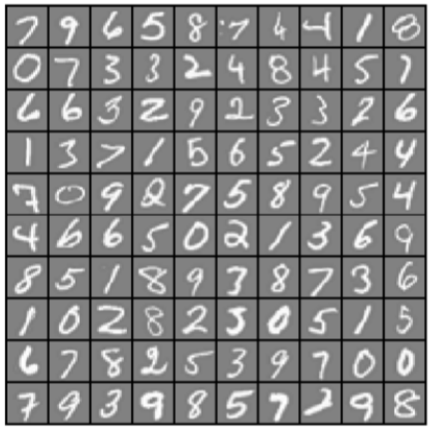
$x_{41} \ \dots \ x_{60}$

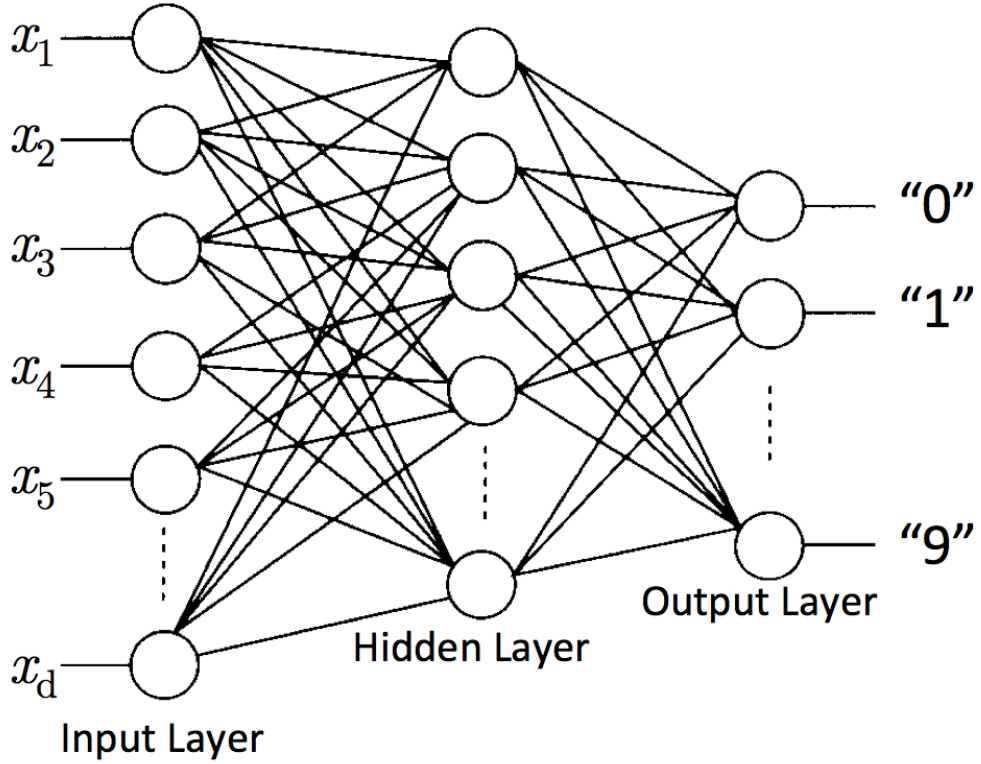
$\vdots$

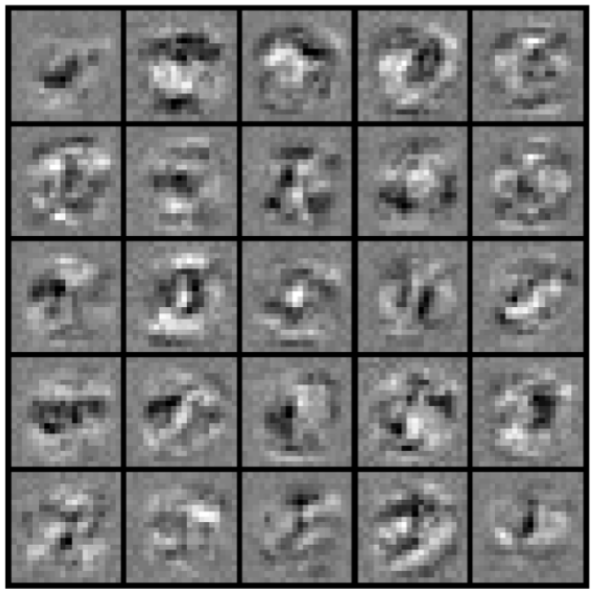
$x_{381} \ \dots \ x_{400}$

Каждое изображение “разворачивается” в вектор $\mathbf{x}$ интенсивности пикселей

Представление слоёв II







Visualization of Hidden Layer

1 Мотивация

2 Персептрон

3 Нейронные сети

4 Обратное распространение ошибки

Обучение нейросетей: обратное распространение

- Похоже на алгоритм обучения персептрона, мы итерируемся по нашим примерам
  - Если выход нейросети корректен, то изменений не производим
  - Если ошибка, веса обновляются для уменьшения ошибки
- Хитрость заключается в том, чтобы оценить вклад в ошибку и разделить ее между способствующими весами

Функция потерь: Log-Loss

Логистическая регрессия для бинарного $\{0,1\}$-случая:

$$\widehat{R}(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y_i \log h_\theta(\mathbf{x}_i) + (1-y_i)\log(1-h_\theta(\mathbf{x}_i))\right] + \frac{\lambda}{2m}\sum_{j=1}^{N}\theta_j^2$$

Нейронная сеть:

$$h_\theta \in \mathbb{R}^K, \quad (h_\theta(\mathbf{x}))_k = k^{\text{th}} \text{ выход}$$

$$\widehat{R}(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_{ik}\log(h_\theta(\mathbf{x}_i))_k + (1-y_{ik})\log\left(1-(h_\theta(\mathbf{x}_i))_k\right)\right]$$
$$+ \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l-1}\sum_{j=1}^{s_l}\left(\theta_{ji}^{(l)}\right)^2$$

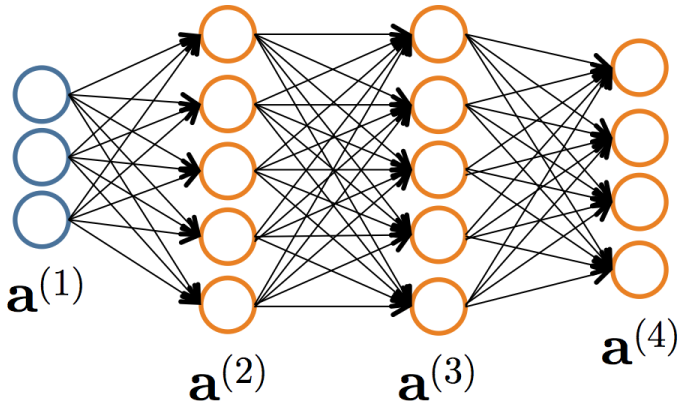
Оптимизация нейронной сети

$$\widehat{R}(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_{ik}\log(h_\theta(\mathbf{x}_i))_k + (1-y_{ik})\log\left(1-(h_\theta(\mathbf{x}_i))_k\right)\right]$$
$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l-1}\sum_{j=1}^{s_l}\left(\theta_{ji}^{(l)}\right)^2$$

- Минимизируем $\widehat{R}(\theta)$
- Нужен код для вычисления
  - $\widehat{R}(\theta)$
  - $\frac{\partial}{\partial\theta_{ji}^{(l)}}\widehat{R}(\theta)$
- $\widehat{R}(\theta)$ не выпуклая, поэтому градиентный спуск для нейронных сетей сходится в локальный оптимум, однако на практике это работает хорошо

Прямое распространение

- Задан один размеченный тренировочный пример $(\mathbf{x}, y)$



<u>Forward Propagation</u>

- $\mathbf{a}^{(1)} = \mathbf{x}$
- $\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{a}^{(1)}$
- $\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$ [add $a_0^{(2)}$]
- $\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$
- $\mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$ [add $a_0^{(3)}$]
- $\mathbf{z}^{(4)} = \Theta^{(3)}\mathbf{a}^{(3)}$
- $\mathbf{a}^{(4)} = h_\Theta(\mathbf{x}) = g(\mathbf{z}^{(4)})$

Интуиция обратного распространения I

- Каждый скрытый нейрон j "ответственнен" за часть ошибки $\delta_j^{(l)}$ в каждом выходном нейроне, до которого есть связь
- $\delta_j^{(l)}$ разделён в соответствии с силой связи между скрытым и выходным нейронами
- Затем, "ошибка" распространяется обратно для того, чтобы предоставить ошибку от скрытых слоёв

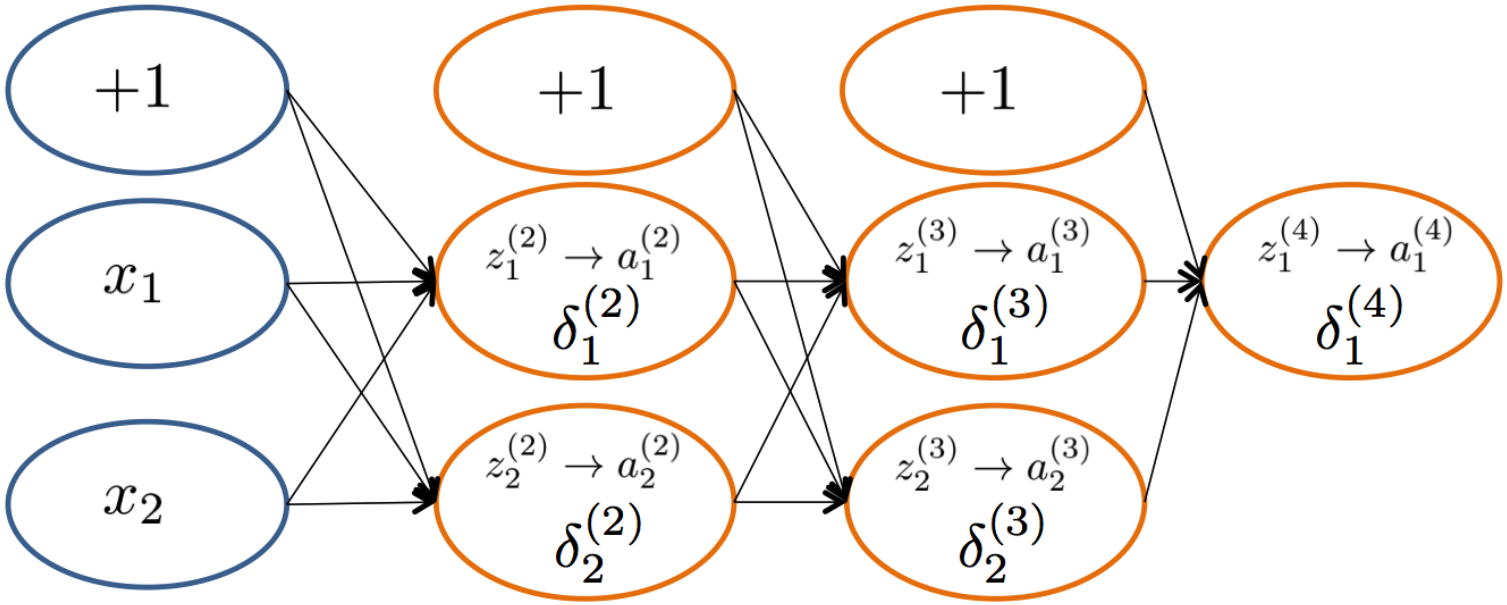
Интуиция обратного распространения I

- Каждый скрытый нейрон j "ответственнен" за часть ошибки $\delta_j^{(l)}$ в каждом выходном нейроне, до которого есть связь
- $\delta_j^{(l)}$ разделён в соответствии с силой связи между скрытым и выходным нейронами
- Затем, "ошибка" распространяется обратно для того, чтобы предоставить ошибку от скрытых слоёв

Интуиция обратного распространения I

- Каждый скрытый нейрон j “ответственнен” за часть ошибки $\delta_j^{(l)}$ в каждом выходном нейроне, до которого есть связь
- $\delta_j^{(l)}$ разделён в соответствии с силой связи между скрытым и выходным нейронами
- Затем, “ошибка” распространяется обратно для того, чтобы предоставить ошибку от скрытых слоёв

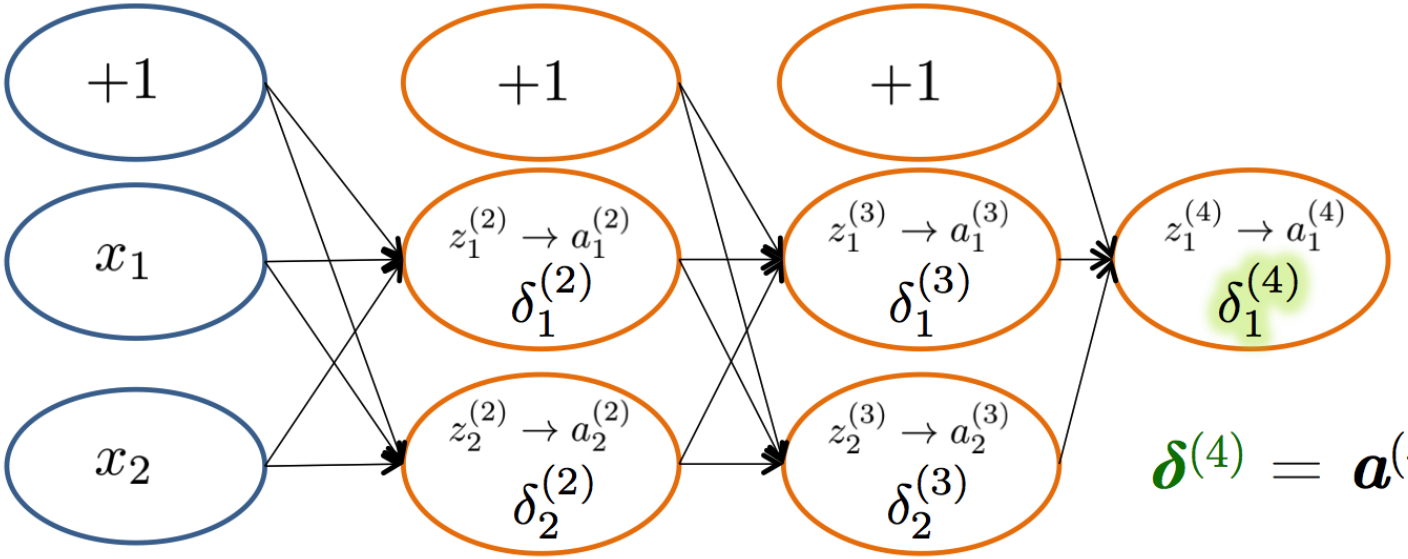
Интуиция обратного распространения II



- $\delta_j^{(l)}$ = "ошибка" нейрона j в слое l
- Формально, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \mathrm{cost}(\mathbf{x}_i)$,

  где $\mathrm{cost}(\mathbf{x}_i) = y_i \log h_\theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\theta(\mathbf{x}_i))$

Интуиция обратного распространения III

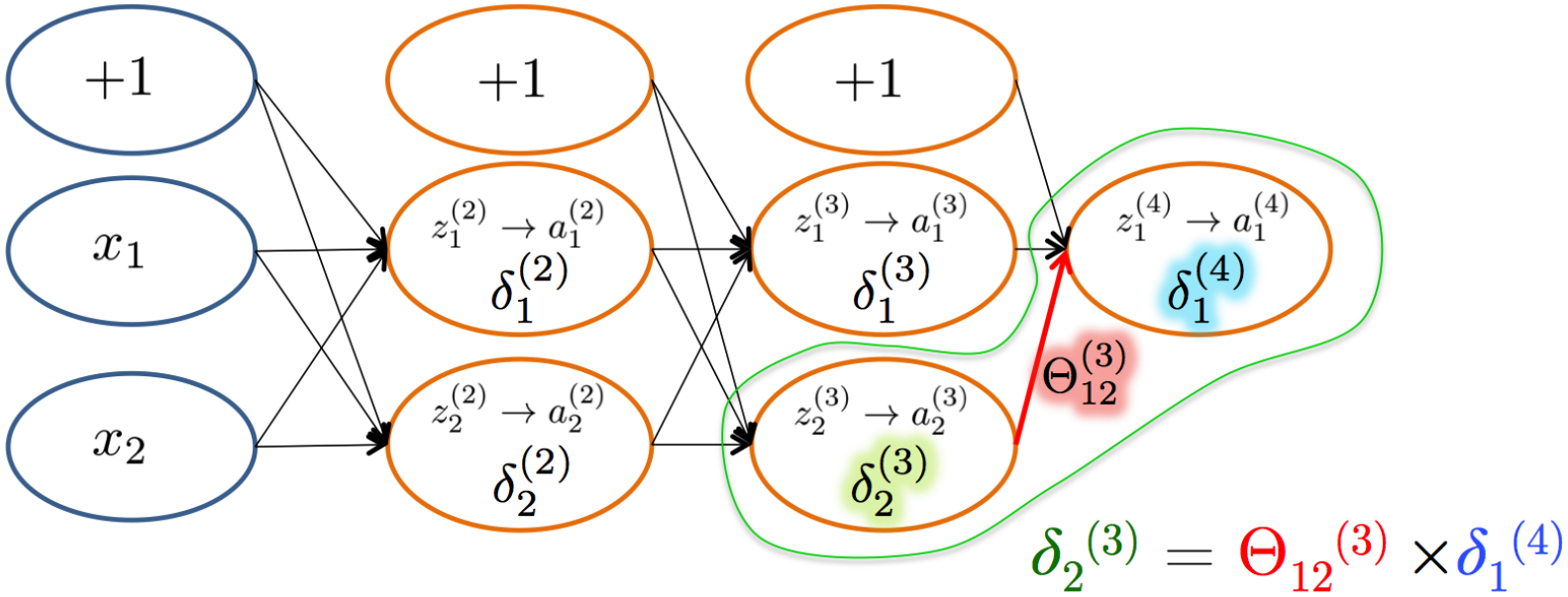


$$\boldsymbol{\delta}^{(4)} = \boldsymbol{a}^{(4)} - \mathbf{y}$$

- $\delta_j^{(l)}$ = "ошибка" нейрона j в слое l
- Формально, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \mathrm{cost}(\mathbf{x}_i)$,

  где $\mathrm{cost}(\mathbf{x}_i) = y_i \log h_\theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\theta(\mathbf{x}_i))$

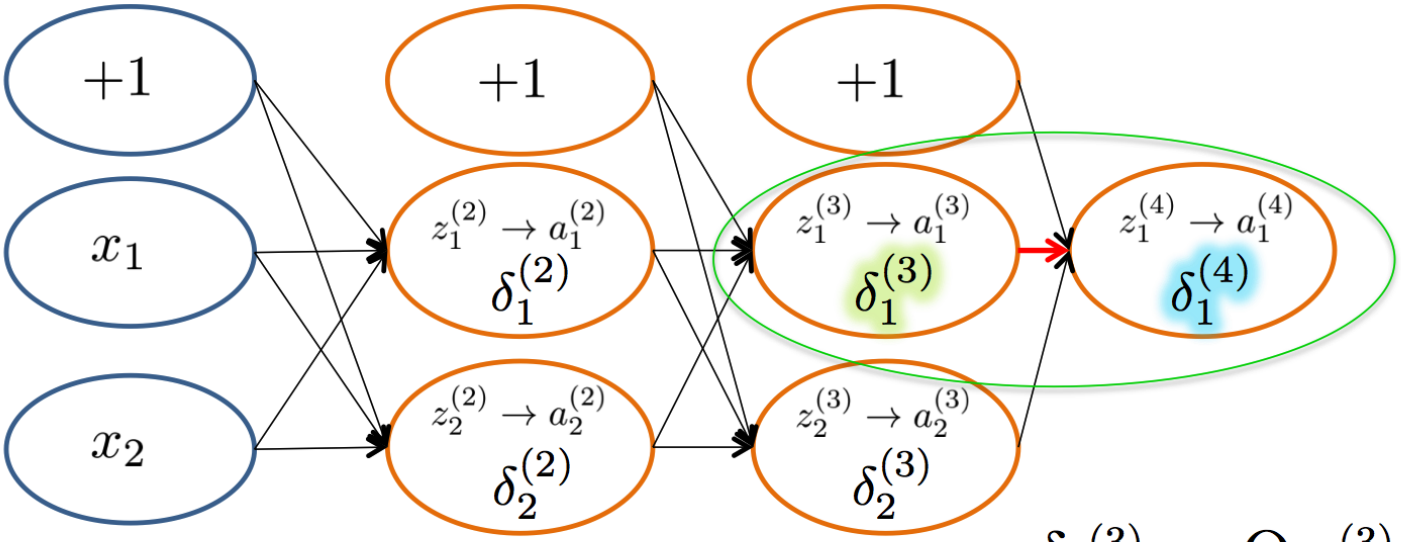
Интуиция обратного распространения IV



- $\delta_j^{(l)}$ = "ошибка" нейрона j в слое l
- Формально, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \mathrm{cost}(\mathbf{x}_i)$,

  где $\mathrm{cost}(\mathbf{x}_i) = y_i \log h_\theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\theta(\mathbf{x}_i))$

Интуиция обратного распространения V



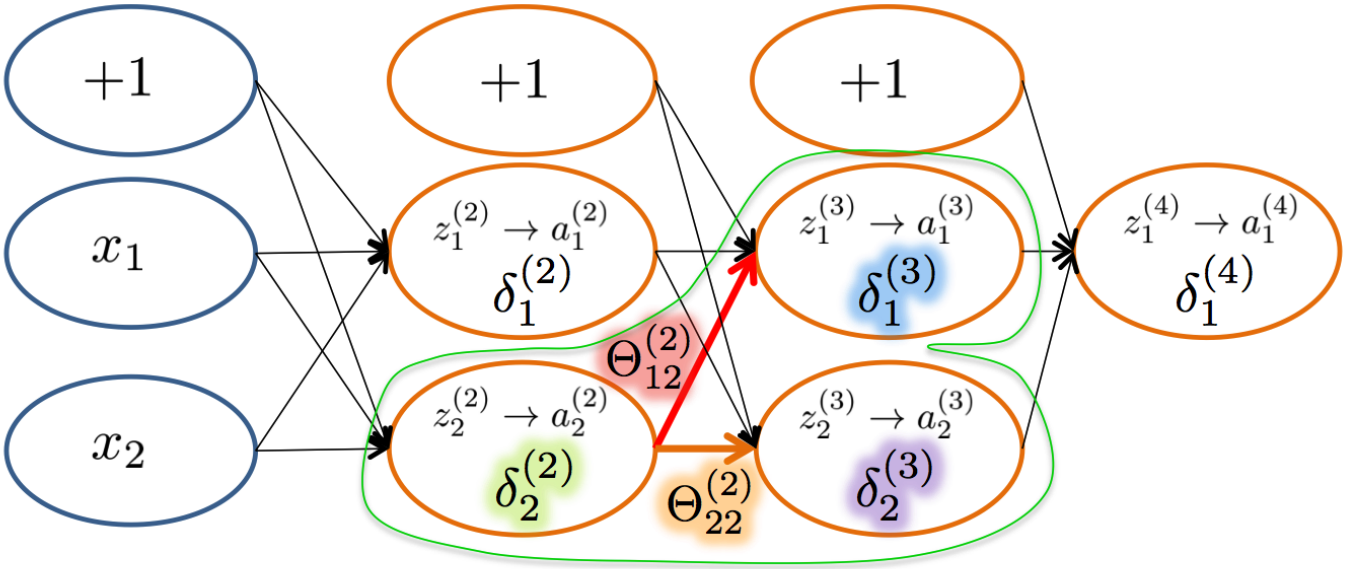
$$\delta_2{}^{(3)} = \Theta_{12}{}^{(3)} \times \delta_1{}^{(4)}$$

$$\delta_1{}^{(3)} = \Theta_{11}{}^{(3)} \times \delta_1{}^{(4)}$$

- $\delta_j^{(l)}$ = "ошибка" нейрона j в слое l
- Формально, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \mathrm{cost}(\mathbf{x}_i)$,

  где $\mathrm{cost}(\mathbf{x}_i) = y_i \log h_\theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\theta(\mathbf{x}_i))$

Интуиция обратного распространения VI



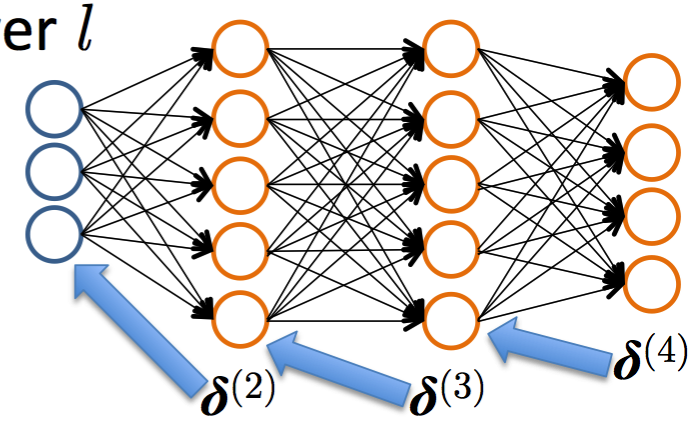
$$\delta_2{}^{(2)} = \Theta_{12}{}^{(2)} \times \delta_1{}^{(3)} + \Theta_{22}{}^{(2)} \times \delta_2{}^{(3)}$$

- $\delta_j^{(l)}$ = "ошибка" нейрона j в слое l
- Формально, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \mathrm{cost}(\mathbf{x}_i)$,

  где $\mathrm{cost}(\mathbf{x}_i) = y_i \log h_\theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\theta(\mathbf{x}_i))$

Обратное распространение: вычисления градиента

Let $\delta_j^{(l)}$ = "error" of node j in layer l



(#layers L = 4)

Element-wise product .*

Backpropagation

- $\boldsymbol{\delta}^{(4)} = \boldsymbol{a}^{(4)} - \mathbf{y}$
- $\boldsymbol{\delta}^{(3)} = (\Theta^{(3)})^\mathsf{T}\boldsymbol{\delta}^{(4)} .* \boxed{g'(\mathbf{z}^{(3)})}$
- $\boldsymbol{\delta}^{(2)} = (\Theta^{(2)})^\mathsf{T}\boldsymbol{\delta}^{(3)} .* \boxed{g'(\mathbf{z}^{(2)})}$
- (No $\boldsymbol{\delta}^{(1)}$)

$g'(\mathbf{z}^{(3)}) = \mathbf{a}^{(3)} .* (1 - \mathbf{a}^{(3)})$

$g'(\mathbf{z}^{(2)}) = \mathbf{a}^{(2)} .* (1 - \mathbf{a}^{(2)})$

$$\frac{\partial}{\partial \theta_{ij}^{(l)}} \widehat{R}(\theta) = a_j^{(l)} \delta_i^{(l+1)}$$ (игнорируя λ; если $\lambda = 0$)

Обратное распространение

Set $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$ (Used to accumulate gradient)

For each training instance $(\mathbf{x}_i, y_i)$:

Set $\mathbf{a}^{(1)} = \mathbf{x}_i$

Compute $\{\mathbf{a}^{(2)}, \ldots, \mathbf{a}^{(L)}\}$ via forward propagation

Compute $\boldsymbol{\delta}^{(L)} = \mathbf{a}^{(L)} - y_i$

Compute errors $\{\boldsymbol{\delta}^{(L-1)}, \ldots, \boldsymbol{\delta}^{(2)}\}$

Compute gradients $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

Compute avg regularized gradient $D_{ij}^{(l)} = \begin{cases} \frac{1}{m}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{m}\Delta_{ij}^{(l)} & \text{otherwise} \end{cases}$

- $\mathbf{D}^{(l)}$ — матрица частных производных $\widehat{R}(\theta)$
- Замечание: можем векторизовать $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ как $\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)} (\mathbf{a}^{(l)})^{\mathrm{T}}$

Обучение нейросети с помощью Градиентного спуска и обратного распространения

Given: training set $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$
Initialize all $\Theta^{(l)}$ randomly (NOT to 0!)
Loop // each iteration is called an epoch

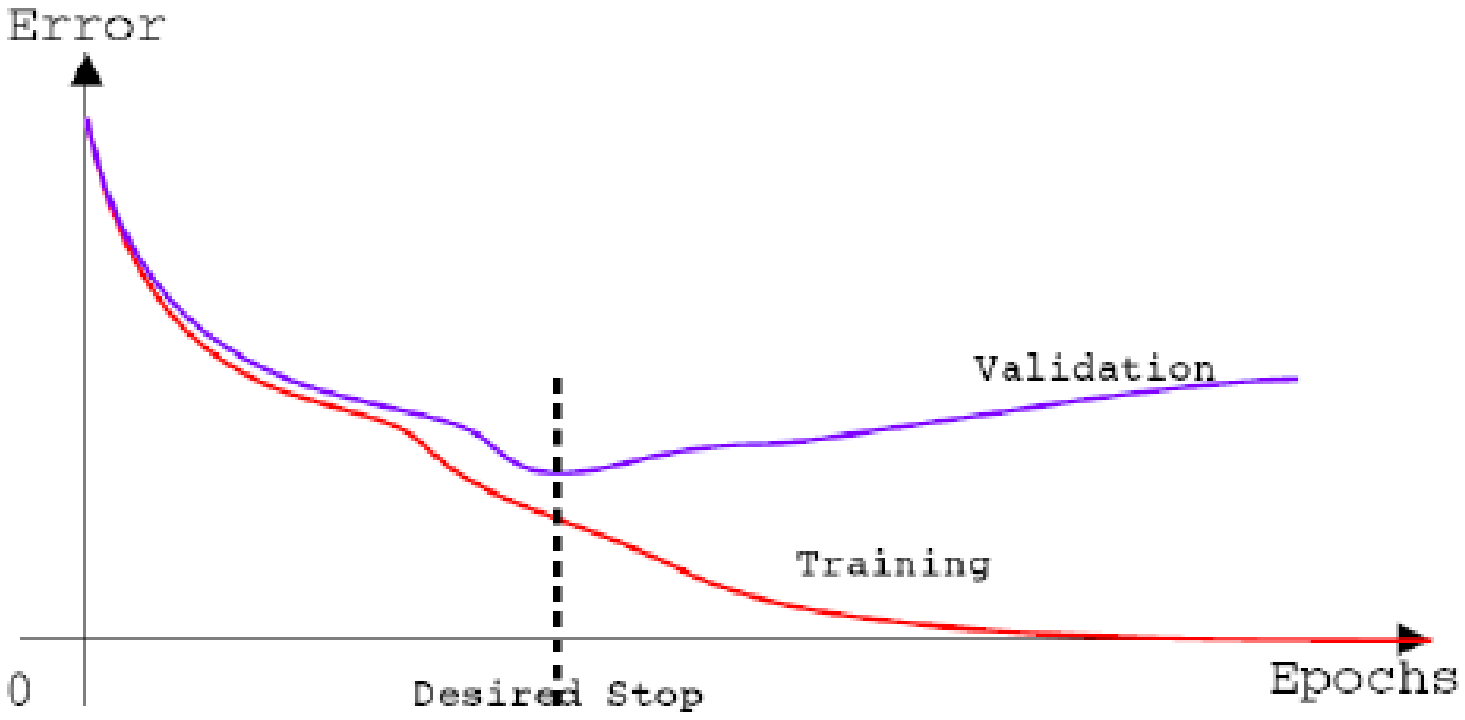
Set $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$ (Used to accumulate gradient)
For each training instance $(\mathbf{x}_i, y_i)$:
Set $\mathbf{a}^{(1)} = \mathbf{x}_i$
Compute $\{\mathbf{a}^{(2)}, \ldots, \mathbf{a}^{(L)}\}$ via forward propagation
Compute $\boldsymbol{\delta}^{(L)} = \mathbf{a}^{(L)} - y_i$
Compute errors $\{\boldsymbol{\delta}^{(L-1)}, \ldots, \boldsymbol{\delta}^{(2)}\}$
Compute gradients $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

Compute avg regularized gradient $D_{ij}^{(l)} = \begin{cases} \frac{1}{m}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{m}\Delta_{ij}^{(l)} & \text{otherwise} \end{cases}$

Update weights via gradient step $\Theta_{ij}^{(l)} = \Theta_{ij}^{(l)} - \alpha D_{ij}^{(l)}$
Until weights converge or max #epochs is reached

Вопросы, связанные с обратным распространением

- Локальный минимум
- Случайная инициализация
- Случайное разделение на данные обучения и валидации
- Оптимизационные методы второго порядка
- Ранняя остановка



Персептрон = стохастический градиентный спуск

- **Целевая функция**: выпуклая, но недифференцируема

$$F(\mathbf{w}) = \frac{1}{T}\sum_{t=1}^{T} \max\left(0, -y_t(\mathbf{w}\cdot\mathbf{x}_t)\right) = \mathbb{E}_{\mathbf{x}\sim\widehat{D}}[f(\mathbf{w},\mathbf{x})]$$

  с $f(\mathbf{w},\mathbf{x}) = \max(0, -y(\mathbf{w}\cdot\mathbf{x}))$

- **Стохастический градиент**: для всех $\mathbf{x}_t$, обновление имеет вид

$$\mathbf{w}_{t+1} \leftarrow \begin{cases} \mathbf{w}_t - \eta\nabla_{\mathbf{w}} f(\mathbf{w}_t, \mathbf{x}_t) & \text{если дифференцируемо,} \\ \mathbf{w}_t & \text{иначе,} \end{cases}$$

  где η — параметр скорости обучения (learning rate)

- Здесь:

$$\mathbf{w}_{t+1} \leftarrow \begin{cases} \mathbf{w}_t + \eta y_t \mathbf{x}_t & \text{if } y_t(\mathbf{w}_t\cdot\mathbf{x}_t) < 0 \\ \mathbf{w}_t & \text{иначе} \end{cases}$$

w

Следующая теорема дает маргинальную верхнюю оценку числа ошибок или обновлений, сделанных алгоритмом Персептрона при обработке последовательности точек T, которые могут быть линейно разделены гиперплоскостью с отступом $\rho > 0$.

- **Теорема**: Пусть $\mathbf{x}_1, \ldots, \mathbf{x}_T \in \mathbb{R}^N$ последовательность T точек с $\|\mathbf{x}_t\| \leq R$ для всех $t \in [1, T]$, для некоторого $R > 0$. Предположим, что существует $\rho > 0$ и $\mathbf{v} \in \mathbb{R}^N$ такое, что для всех $t \in [1, T]$, $\rho \leq \frac{y_t(\mathbf{v} \cdot \mathbf{x}_t)}{\|\mathbf{v}\|}$. Тогда количество обновлений ($=$ количество ошибок), сделанных алгоритмом Персептрона при обработке $\mathbf{x}_1, \ldots, \mathbf{x}_T \in \mathbb{R}^N$ ограничена $\frac{R^2}{\rho^2}$.

Комментарии

- граница не зависит от размерности и tight
- Сходимость может быть медленная при маленьком отступе, количество обновлений, сделанных алгоритмом, может быть в $\Omega(2^N)$
- Среди прочих вариантов: алгоритм голосования для персептрона. Предсказывает в соответствии с

$$\operatorname{sign}\left(\left(\sum_{t\in I} c_t \mathbf{w}_t\right)\cdot \mathbf{x}\right),$$

  где c_t — количество обновлений $\mathbf{w}_t$
- $\{\mathbf{x}_t : t \in I\}$ — опорные вектора для алгоритма персептрона, так как $\mathbf{w}_T = \sum_{t\in I} y_t \mathbf{x}_t$
- Неразделимый случай: отсутствует сходимость