ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ

Лекция 3

Многослойный перцептрон (MLP)

Архитектура, скрытые слои, нелинейность, матричное представление и прямое распространение сигнала

Преподаватель: Аршын Алтыбай

Кафедра компьютерной инженерии,
Алматинский технологический университет, Алматы, Казахстан

Учебный год: 2026

> **Место в курсе.** На лекции 2 мы построили пороговый линейный классификатор и доказали невозможность разделить XOR одной гиперплоскостью. Теперь изучим конструкцию из последовательно соединённых нейронных слоёв и выясним, каким образом *нелинейная активация* снимает это ограничение. В этой лекции основное внимание уделяется *прямому распространению*; обучение и back-propagation рассматриваются как следующий шаг [6, 7].

1. Цели и ожидаемые результаты

Цель лекции: сформировать точное понимание структуры полносвязной многослойной нейронной сети, её вычислений и математических возможностей; научить рассчитывать MLP вручную и программно.

После лекции обучающийся сможет:

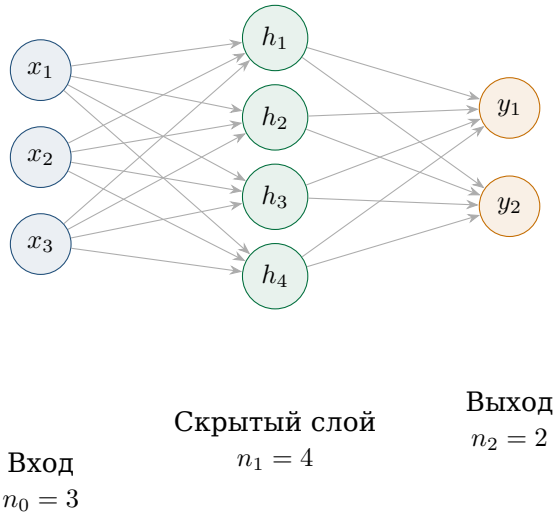
1. отличать однослойный перцептрон от многослойного и объяснять функцию скрытого слоя;
2. записывать $z^{(\ell)} = W^{(\ell)} a^{(\ell-1)} + b^{(\ell)}$ и $a^{(\ell)} = \varphi_\ell(z^{(\ell)})$;
3. определять размерности тензоров, вычислять количество обучаемых параметров и стоимость прямого прохода;
4. объяснять принципиальную роль нелинейной активации;
5. вручную проводить прямой проход для сети небольшого размера;
6. различать выходные слои для регрессии, бинарной и многоклассовой классификации;
7. конструировать MLP для XOR и проверять его результаты;

8. реализовывать forward pass на NumPy и сопоставлять его с библиотечной моделью.

2. От одного нейрона к многослойной структуре

Одиночный искусственный нейрон принимает признаки $x \in \mathbb{R}^d$, вычисляет $z = w^\top x + b$ и преобразует его посредством активации $a = \varphi(z)$. Если выход бинарный и определяется знаком z, граница классификации остаётся линейной. Однослойные классификаторы хорошо моделируют простые зависимости, но не способны строить нелинейно разделимые границы, например XOR [1, 2].

Многослойный перцептрон (multilayer perceptron, MLP) — нейронная сеть прямого распространения, в которой информация последовательно проходит от входа через один или несколько скрытых полносвязных слоёв к выходу. В классическом MLP каждое число векторного представления следующего слоя вычисляется из всех активаций предыдущего слоя. Термин *перцептрон* в названии MLP является историческим: современные MLP обычно используют гладкие или кусочно-линейные функции активации, а не классическую пороговую функцию.



Терминологическое уточнение Входные признаки иногда изображают в виде «входных нейронов». Обычно это только компоненты входного вектора, а не обучаемый слой с весами. Поэтому число обучаемых аффинных преобразований в сети с одним скрытым слоем равно двум.

3. Слои, индексы, параметры и обозначения

Пусть в сети L обучаемых слоёв, не считая входного. Число элементов входного вектора $n_0 = d$, ширина слоя ℓ равна n_ℓ. Введём

$$a^{(0)} = x \in \mathbb{R}^{n_0}, \qquad W^{(\ell)} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}, \qquad b^{(\ell)} \in \mathbb{R}^{n_\ell}.$$

Для $\ell = 1, \dots, L$ вычисления определяются рекуррентно:

$$z^{(\ell)} = W^{(\ell)} a^{(\ell-1)} + b^{(\ell)}, \qquad a^{(\ell)} = \varphi_\ell\big(z^{(\ell)}\big). \tag{1}$$

Функция φ_ℓ обычно действует поэлементно. В выходном слое она может быть отличной от активации скрытых слоёв. Обозначим все обучаемые параметры через $\theta = \{W^{(\ell)}, b^{(\ell)}\}_{\ell=1}^{L}$; тогда вся сеть представляет параметризованное отображение $f_\theta : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$.

3.1 Компонентная запись

j-й нейрон слоя ℓ вычисляет

$$z_j^{(\ell)} = \sum_{i=1}^{n_{\ell-1}} W_{ji}^{(\ell)} a_i^{(\ell-1)} + b_j^{(\ell)}, \qquad a_j^{(\ell)} = \varphi_\ell(z_j^{(\ell)}). \tag{2}$$

Первый индекс j указывает получающий нейрон, а второй i — отправляющий. Если $W^{(1)}$ имеет размер 4×3, то этот слой превращает три входа в четыре активации. В литературе и коде возможны и другие соглашения об ориентации матриц: формулы необходимо согласовывать с выбранным соглашением.

3.2 Почему смещение — не «вес ещё одного признака» буквально

Действительно, b можно алгебраически включить в матрицу весов, дополнив вход единицей:

$$\widetilde{a} = \begin{pmatrix} a \\ 1 \end{pmatrix}, \quad \widetilde{W} = \begin{pmatrix} W & b \end{pmatrix}, \quad Wa + b = \widetilde{W}\widetilde{a}.$$

Но в программных библиотеках b чаще хранится отдельно и не умножается на переменную входа. Такая запись упрощает подсчёт параметров и градиентов [6, 7].

4. Композиция отображений и представления признаков

Однослойный классификатор строит одну разделяющую гиперплоскость. MLP последовательно применяет аффинные преобразования и нелинейные отображения:

$$f_\theta(x) = \varphi_L\big(W^{(L)}\varphi_{L-1}\big(\cdots\varphi_1(W^{(1)}x + b^{(1)})\cdots\big) + b^{(L)}\big)\,. \tag{3}$$

Скрытый слой можно интерпретировать как *выученное представление*: исходные координаты превращаются в признаки, по которым последующие слои строят решение. Представления не обязательно имеют непосредственный физический смысл; их полезность определяется задачей и результатом обучения [4, 6].

4.1 Почему нелинейность необходима: точное алгебраическое доказательство

Рассмотрим два последовательных слоя *без* нелинейной активации:

$$a^{(1)} = W^{(1)}x + b^{(1)}, \qquad a^{(2)} = W^{(2)}a^{(1)} + b^{(2)}.$$

Подставим первое равенство во второе:

$$\begin{aligned} a^{(2)} &= W^{(2)}\big(W^{(1)}x + b^{(1)}\big) + b^{(2)} \\ &= \underbrace{W^{(2)}W^{(1)}}_{W_{\text{eff}}}x + \underbrace{\big(W^{(2)}b^{(1)} + b^{(2)}\big)}_{b_{\text{eff}}}. \end{aligned}$$

Таким образом, два слоя эквивалентны одному аффинному преобразованию. По индукции то же верно для любой конечной последовательности аффинных слоёв без нелинейной активации. **Вывод: увеличение числа линейных слоёв само по себе не превращает линейный классификатор в нелинейный.**

> **Зачем нужен скрытый слой?** Если после каждого скрытого аффинного преобразования стоит нелинейная функция, вычисление в общем случае невозможно свести к единственной матрице и смещению. Именно эта композиционная нелинейность позволяет моделировать сложные зависимости, включая XOR.

5. Функции активации: выбор и свойства

Изучим распространённые функции, применяемые поэлементно в скрытых слоях и на выходе.

Функция	Определение	Применение и ограничения
Sigmoid	$\sigma(z) = 1/(1+e^{-z})$	Выход $(0,1)$; подходит для бинарной вероятностной модели, при больших $\lvert z\rvert$ производная мала.
Tanh	$\tanh(z) = (e^{z}-e^{-z})/(e^{z}+e^{-z})$	Выход $(-1,1)$, центрирован около нуля; насыщается при больших $\lvert z\rvert$.
ReLU	$\max(0, z)$	Дешёвое вычисление; не насыщается справа; при $z<0$ производная равна нулю.
Leaky ReLU	$\max(\alpha z, z)$, $0<\alpha<1$	Сохраняет небольшой наклон на отрицательной полуоси.
Identity	$\varphi(z) = z$	Часто в выходном слое регрессии; как единственная активация скрытых слоёв недостаточна.

Для ReLU производная равна 0 при $z<0$ и 1 при $z>0$; в точке $z=0$ классической производной нет. На практике используется принятое в реализации значение субградиента (часто 0). Для сигмоиды

$$\sigma'(z) = \sigma(z)\big(1-\sigma(z)\big), \qquad 0 < \sigma'(z) \le \frac{1}{4}. \tag{4}$$

Это объясняет, почему при определённых условиях сигмоида может способствовать затуханию градиентов в глубоких сетях. Подробная теория дифференцирования и backpropagation рассматривается отдельно [7, 6].

5.1 Небольшой численный пример

Пусть $z = (-2, 0, 3)^\top$. Тогда

$$\mathrm{ReLU}(z) = (0, 0, 3)^\top, \qquad \sigma(z) \approx (0.1192, 0.5, 0.9526)^\top.$$

Применение активации не меняет число элементов вектора, но существенно меняет геометрию отображения.

6. Прямое распространение на конкретной сети

Рассмотрим сеть с двумя входами, двумя нейронами скрытого слоя и одним выходом:

$$(n_0, n_1, n_2) = (2, 2, 1), \qquad \varphi_1 = \mathrm{ReLU}, \quad \varphi_2 = \sigma.$$

Зададим

$$W^{(1)} = \begin{pmatrix} 1 & -1 \\ 0.5 & 2 \end{pmatrix}, \quad b^{(1)} = \begin{pmatrix} 0 \\ -0.5 \end{pmatrix}, \quad W^{(2)} = \begin{pmatrix} 1 & -2 \end{pmatrix}, \quad b^{(2)} = 0.2.$$

Шаг 1. Для $x = (2, 1)^\top$ вычисляем

$$z^{(1)} = \begin{pmatrix} 1 \cdot 2 + (-1) \cdot 1 \\ 0.5 \cdot 2 + 2 \cdot 1 - 0.5 \end{pmatrix} = \begin{pmatrix} 1 \\ 2.5 \end{pmatrix}.$$

Шаг 2. Применяем ReLU:

$$a^{(1)} = \mathrm{ReLU}\left(\begin{pmatrix} 1 \\ 2.5 \end{pmatrix} \right) = \begin{pmatrix} 1 \\ 2.5 \end{pmatrix}.$$

Шаг 3. Выходной слой:

$$z^{(2)} = \begin{pmatrix} 1 & -2 \end{pmatrix} \begin{pmatrix} 1 \\ 2.5 \end{pmatrix} + 0.2 = 1 - 5 + 0.2 = -3.8.$$

Шаг 4. Сигмоида:

$$a^{(2)} = \sigma(-3.8) = \frac{1}{1 + e^{3.8}} \approx 0.021881.$$

Если это бинарная классификация с порогом 0.5, предсказан отрицательный класс (0). Здесь 0.021881 — *выход модели*; вероятность следует считать калиброванной только при дополнительных предположениях и проверке качества [3].

> **Проверка размерностей** $W^{(1)}$ имеет размер 2×2, $a^{(0)}$ — 2×1, поэтому $z^{(1)}$ — 2×1. $W^{(2)}$ имеет размер 1×2, поэтому $z^{(2)}$ — скаляр. Несовместимость размерностей — одна из самых частых ошибок начинающих.

7. Количество параметров и вычислительная стоимость

Для одного полносвязного слоя с $n_{\ell-1}$ входами и n_ℓ нейронами нужно $n_\ell n_{\ell-1}$ весов и n_ℓ смещений. Следовательно,

$$P = \sum_{\ell=1}^{L} (n_\ell n_{\ell-1} + n_\ell) = \sum_{\ell=1}^{L} n_\ell (n_{\ell-1} + 1). \tag{5}$$

Пример: MLP с архитектурой $4 \to 8 \to 3 \to 1$ содержит

$$P = 8(4+1) + 3(8+1) + 1(3+1) = 40 + 27 + 4 = 71$$

обучаемый параметр. Для нашей сети $2 \to 2 \to 1$ их $2(2+1) + 1(2+1) = 9$.

Одна матрично-векторная операция слоя требует порядка $O(n_{\ell-1} n_\ell)$ умножений/сложений. Полный прямой проход имеет порядок

$$O\left(\sum_{\ell=1}^{L} n_{\ell-1} n_\ell \right),$$

если стоимость простых поэлементных активаций считается линейной по ширине слоя. Это асимптотическая оценка, а не точное измерение времени работы.

7.1 Глубина и ширина

Ширина — количество нейронов в слое; *глубина* обычно измеряется числом обучаемых слоёв. Большая ширина повышает потенциальную выразительность, но увеличивает стоимость и риск переобучения при недостатке данных. Большая глубина позволяет строить многоступенчатые представления, но осложняет оптимизацию. Правильный выбор архитектуры определяется задачей и экспериментами, а не принципом «чем больше, тем лучше» [6, 4].

8. Несколько объектов сразу: матричная форма batch inference

В предыдущих формулах x — один вектор-столбец. Для m объектов объединим входы в матрицу

$$A^{(0)} = X = \begin{pmatrix} | & & | \\ x^{[1]} & \cdots & x^{[m]} \\ | & & | \end{pmatrix} \in \mathbb{R}^{n_0 \times m}.$$

Тогда прямой проход слоя записывается как

$$Z^{(\ell)} = W^{(\ell)} A^{(\ell-1)} + b^{(\ell)} \mathbf{1}_m^\top, \qquad A^{(\ell)} = \varphi_\ell(Z^{(\ell)}). \tag{6}$$

Вектор $b^{(\ell)}$ добавляется к каждому столбцу. Это и есть *broadcasting* на уровне математической формулы. В NumPy часто предпочитают хранить объекты *в строках*: тогда эквивалентная формула выглядит как $Z = AW^\top + b$ с $A \in \mathbb{R}^{m \times n_{\ell-1}}$. Обе конвенции корректны, если сохранять согласованность.

9. Как MLP реализует XOR: конструктивное решение

Рассмотрим $x_1, x_2 \in \{0, 1\}$ и желаемый выход $x_1 \oplus x_2$. Двух скрытых нейронов с ReLU достаточно, чтобы построить точный ответ на четырёх вершинах булева квадрата:

$$h_1 = \mathrm{ReLU}(x_1 - x_2), \tag{7}$$
$$h_2 = \mathrm{ReLU}(x_2 - x_1), \tag{8}$$
$$y = h_1 + h_2 = |x_1 - x_2|. \tag{9}$$

Для входов $(0, 0)$ и $(1, 1)$ результат равен 0, для $(0, 1)$ и $(1, 0)$ — 1. В матричной записи:

$$W^{(1)} = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \quad b^{(1)} = 0, \quad W^{(2)} = \begin{pmatrix} 1 & 1 \end{pmatrix}, \quad b^{(2)} = 0.$$

x_1	x_2	h_1	h_2	**Выход**
0	0	0	0	0
0	1	0	1	1
1	0	1	0	1
1	1	0	0	0

Смысл примера. Скрытый слой переводит исходную точку в новое представление $h = (h_1, h_2)$, в котором нужный выход вычисляется линейной комбинацией. Сеть не «нарушает» теорему о невозможности XOR для одной линейной границы: она использует *нелинейное преобразование входа*.

Заметим, что приведённый выход на *всём* квадрате $[0, 1]^2$ равен $|x_1 - x_2|$. Он совпадает с булевой функцией XOR на четырёх двоичных вершинах, но не задаёт бинарную метку для произвольных вещественных входов без дополнительного правила принятия решения.

10. Выходной слой зависит от задачи

Последний слой MLP не всегда устроен так же, как скрытые слои. Рассмотрим три стандартных варианта [3, 4].

10.1 Регрессия

Для непрерывной целевой переменной $y \in \mathbb{R}$ часто используется линейный выход $\hat{y} = z^{(L)}$ и квадратичная функция потерь

$$\mathcal{L}_{\text{MSE}} = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2.$$

Например, прогнозировать энергопотребление, температуру или стоимость. При ограниченном диапазоне выхода может использоваться другое преобразование.

10.2 Бинарная классификация

Для двух классов: один выходной нейрон с сигмоидой $p = \sigma(z)$ и бинарная кросс-энтропия

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{m} \sum_{i=1}^{m} \left[y_i \log p_i + (1 - y_i) \log(1 - p_i) \right].$$

Для устойчивых вычислений библиотеки обычно объединяют логиты и loss в одной функции (например, BCEWithLogitsLoss в PyTorch), чтобы избежать проблем точности при p близком к нулю или единице.

10.3 Многоклассовая классификация

Если классов $K \geq 3$ и они взаимоисключающие, используются K выходных логитов z_k и softmax:

$$p_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}, \qquad \sum_{k=1}^{K} p_k = 1. \tag{10}$$

Для численной устойчивости вычисляют $\exp(z_k - \max_j z_j)$. Основная функция потерь — многоклассовая кросс-энтропия. При *многометочной* классификации, когда одному объекту могут соответствовать несколько классов, используют независимые сигмоиды и соответствующую бинарную loss, а не обычную softmax.

11. Реализация прямого прохода на чистом NumPy

Приведённая программа воспроизводит предыдущий ручной расчёт. Ориентация массивов соответствует формуле $z = Wa + b$.

Пример 1. Один объект, два слоя и контроль размерностей

```python
import numpy as np

x = np.array([2.0, 1.0])
W1 = np.array([[1.0, -1.0], [0.5, 2.0]])
b1 = np.array([0.0, -0.5])
W2 = np.array([[1.0, -2.0]])
b2 = np.array([0.2])

def relu(z):
    return np.maximum(0.0, z)

def sigmoid(z):
    return 1.0 / (1.0 + np.exp(-z))

z1 = W1 @ x + b1
h = relu(z1)
z2 = W2 @ h + b2
p = sigmoid(z2)
print("z1:", z1)
print("h:", h)
print("z2:", z2)
print("p:", np.round(p, 6))
print("parameters:", W1.size + b1.size + W2.size + b2.size)
```

Ожидаемый вывод

```
z1: [1.  2.5]
h: [1.  2.5]
z2: [-3.8]
p: [0.021881]
parameters: 9
```

11.1 Пакет объектов (batch)

Здесь каждая строка X соответствует одному объекту; поэтому веса умножаются с транспонированием.

Пример 2. Векторизация по объектам

```python
X = np.array([[2., 1.], [1., 1.], [0., 0.]])
Z1 = X @ W1.T + b1     # (3, 2)
H = relu(Z1)            # (3, 2)
Z2 = H @ W2.T + b2      # (3, 1)
P = sigmoid(Z2)
print("shape:", P.shape)
print(np.round(P.ravel(), 6))
```

Ожидаемый вывод

```
shape: (3, 1)
[0.021881 0.021881 0.549834]
```

12. XOR в NumPy: воспроизводимый тест

Пример 3. Нелинейный скрытый слой решает XOR

```
X = np.array([[0., 0.], [0., 1.],
              [1., 0.], [1., 1.]])
W1 = np.array([[1., -1.], [-1., 1.]])
b1 = np.zeros(2)
W2 = np.array([[1., 1.]])
b2 = np.zeros(1)
H = np.maximum(0., X @ W1.T + b1)
y = H @ W2.T + b2
print(y.ravel().astype(int).tolist())
```

Ожидаемый вывод

```
[0, 1, 1, 0]
```

Этот код *не обучает* сеть: все веса заданы вручную. В реальных задачах значения $W^{(\ell)}$ и $b^{(\ell)}$ подбираются по данным путём минимизации функции потерь. Это будет предметом последующих лекций.

13. Как готовят MLP к обучению

Стандартный цикл разработки нейросети включает: подготовку данных, выбор архитектуры, инициализацию весов, определение функции потерь, прямой проход, расчёт градиентов, обновление параметров, оценку на независимых данных [3, 5]. В этой лекции изучены архитектура и прямой проход; производные по слоям пока не вычисляются.

13.1 Инициализация параметров

Запуск сети с полностью одинаковыми весами всех нейронов одного слоя нарушает полезную симметрию обучения: при одинаковых входах, активациях и градиентах они могут продолжать учить одинаковую функцию. Случайная инициализация помогает избежать такой симметрии. Популярны схемы Xavier/Glorot и He/Kaiming, выбираемые с учётом

типа активации. Их точный вывод требует анализа распространения дисперсии и изучается позднее.

13.2 Нормализация признаков

Неоднородные масштабы x_j могут осложнять оптимизацию. Для числовых признаков часто используют стандартизацию $x'_j = (x_j - \mu_j)/s_j$, где параметры μ_j, s_j оцениваются *только на обучающей выборке*. Перенос статистики с тестовой выборки в обучение является утечкой информации. Для изображений и категориальных признаков используются соответствующие схемы предобработки [3].

14. MLP как универсальный аппроксиматор: что можно и нельзя утверждать

Классические результаты об универсальной аппроксимации утверждают, что при соответствующих условиях на активацию и на компактном множестве достаточно широкий односкрытослойный feedforward-network может сколь угодно точно аппроксимировать непрерывную функцию. Важно понимать три ограничения такого утверждения:

1. оно говорит о *существовании* параметров, но не гарантирует, что конкретный алгоритм обучения их найдёт;
2. оно не гарантирует малой сети или умеренной вычислительной стоимости;
3. хорошая аппроксимация на заданной области не гарантирует надёжной *экстраполяции* за пределами обучающих данных.

Здесь мы используем утверждение лишь как концептуальную мотивацию, не доказывая теорему в полном общем виде. Строгое обсуждение предпосылок и вариантов результата см. [6].

15. Практические инженерные ошибки

1. **Неправильные размерности.** Всегда выписывайте $W^{(\ell)} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ и форму batch.
2. **Отсутствие нелинейности.** Цепочка линейных слоёв эквивалентна одному аффинному отображению.
3. **Неверная активация на выходе.** Softmax для взаимоисключающих классов, sigmoid для бинарной/многометочной задачи, линейный выход для базовой регрессии.
4. **Смешение вероятности и логита.** Логит z — ещё не вероятность; преобразование задаётся функцией выхода.
5. **Использование тестовых данных при настройке.** Это приводит к завышенной оценке обобщающей способности.
6. **Игнорирование воспроизводимости.** Необходимо фиксировать seed, версии библиотек, разбиение на выборки и настройки сети.
7. **Оценка только по loss на обучении.** Нужны validation/test и подходящие метрики задачи.

16. Итоги лекции

- MLP — композиция обучаемых аффинных слоёв и нелинейных активаций.
- Математическое описание одного слоя: $z^{(\ell)} = W^{(\ell)} a^{(\ell-1)} + b^{(\ell)}$, $a^{(\ell)} = \varphi_\ell(z^{(\ell)})$.
- Без нелинейности любая глубина сводится к одному аффинному преобразованию.
- ReLU-сеть $2 \to 2 \to 1$ может точно представить XOR на булевых входах.
- Число параметров полносвязного MLP равно $\sum_{\ell=1}^{L} n_\ell(n_{\ell-1}+1)$.
- Выходная активация и функция потерь выбираются по постановке задачи.
- Forward pass можно реализовать обычной матричной алгеброй NumPy.

17. Контрольные вопросы

1. Что понимают под многослойным перцептроном (MLP)?
2. Какова роль входного, скрытого и выходного слоёв?
3. Что означает «полносвязный слой»?
4. Чем активация $a^{(\ell)}$ отличается от предактивации $z^{(\ell)}$?
5. Как выбрать размерность $W^{(\ell)}$?

6. Почему линейные слои без активации можно свести к одному?
7. Каким образом ReLU добавляет нелинейность?
8. Почему функция sigmoid может приводить к затуханию градиента?
9. Чем ReLU отличается от Leaky ReLU?
10. Как вычислить прямой проход MLP вручную?
11. Что означает batch inference?
12. Как различаются соглашения «объекты по строкам» и «объекты по столбцам»?
13. Как вычислить число параметров архитектуры $5 \to 10 \to 4 \to 2$?
14. Какова асимптотическая вычислительная стоимость forward pass?
15. Почему XOR требует нелинейного преобразования признаков?
16. Как устроена явная ReLU-конструкция для XOR?
17. Какой выход применяют при регрессии?
18. Когда применяется сигмоида, а когда softmax?
19. Что такое логит?
20. Почему наличие универсальной аппроксимации не гарантирует успешное обучение?

18. Задания для самостоятельной работы

1. **Ручной расчёт.** Для сети из раздела 6 вычислите $a^{(2)}$ при $x = (1, 1)^\top$ и $x = (0, 0)^\top$. Сравните с NumPy.
2. **Подсчёт параметров.** Найдите количество весов, смещений и общее число параметров для MLP $8 \to 16 \to 8 \to 3$; объясните каждый член суммы.
3. **XOR.** Измените порог классификации для непрерывного выхода ReLU-конструкции XOR и покажите точность на четырёх двоичных входах.
4. **Активации.** Постройте графики sigmoid, tanh, ReLU, Leaky ReLU на отрезке $[-5, 5]$ и опишите область насыщения каждой функции.
5. **Собственный код.** Реализуйте функцию `forward(x, weights, biases, activations)` для произвольного количества слоёв, проверяющую размеры матриц.
6. **Аналитическое задание.** Докажите индукцией, что композиция L аффинных преобразований снова аффинна.

19. Библиография и дальнейшее чтение

По вычислениям «с нуля» рекомендуются [1, 2, 7]; для матричного описания и теории аппроксимации — [6]; для практической реализации — [3, 4, 5].

References

[1] Trask A. W. *Grokking Deep Learning*. Manning Publications, 2019. Русское издание: «Грокаем глубокое обучение». https://www.manning.com/books/grokking-deep-learning.

[2] Rashid T. *Make Your Own Neural Network*. 2016. Русское издание: «Глубокое обучение. Лёгкая теория и практика на Python» (название может варьироваться по изданию). https://makeyourownneuralnetwork.blogspot.com/.

[3] Géron A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 3rd ed., O'Reilly Media, 2022. Русское издание: «Прикладное машинное обучение с помощью Scikit-Learn, Keras и TensorFlow». https://www.oreilly.com/library/view/hands-on-machine-learning/9781098125967/.

[4] Chollet F. *Deep Learning with Python*. 2nd ed., Manning Publications, 2021. Русское издание: «Глубокое обучение на Python». https://www.manning.com/books/deep-learning-with-python-second-edition.

[5] Stevens E., Antiga L., Viehmann T. *Deep Learning with PyTorch*. Manning Publications, 2020. Русское издание: «Глубокое обучение с PyTorch». https://www.manning.com/books/deep-learning-with-pytorch.

[6] Goodfellow I., Bengio Y., Courville A. *Deep Learning*. MIT Press, 2016. Русское издание: «Глубокое обучение». https://www.deeplearningbook.org/.

[7] Nielsen M. A. *Neural Networks and Deep Learning*. 2015. Бесплатный онлайн-учебник: «Нейронные сети и глубокое обучение». http://neuralnetworksanddeeplearning.com/.