**ФГБОУ ВО   
Уфимский университет науки и технологий**

**Кафедра ВМиК**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 100 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 90 |  |  |  |  |  |  |  |  |  |  |  |  |
| 80 |  |  |  |  |  |  |  |  |  |  |  |  |
| 70 |  |  |  |  |  |  |  |  |  |  |  |  |
| 60 |  |  |  |  |  |  |  |  |  |  |  |  |
| 50 |  |  |  |  |  |  |  |  |  |  |  |  |
| 40 |  |  |  |  |  |  |  |  |  |  |  |  |
| 30 |  |  |  |  |  |  |  |  |  |  |  |  |
| 20 |  |  |  |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

Мобильное приложение с нейросетью

**ОТЧЕТ**

**по лабораторной работе**

**по** Программированию  
мобильных приложений

(*наименование дисциплины*)

|  |
| --- |
| Лабораторная работа 4 |
| (обозначение документа) |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Группа |  |  | Фамилия, И., О. | Подпись | Дата | Оценка |
| номер |  |
|  |  |
| Студент | | | Студент\_ФИО |  |  |  |
| Преподаватель | | | Преподаватель\_ФИО |  |  |  |
| Принял | | |  |  |  |  |

**Уфа 2024 г****.**

Содержание

[1 Цель работы 3](#_Toc182589089)

[2 Практическая часть 4](#_Toc182589090)

[2.1 Выбор приложения 4](#_Toc182589091)

[2.2 Выбор предметной области и подготовка обучающей выборки 4](#_Toc182589092)

[2.3 Реализация работы многослойного персептрона 8](#_Toc182589093)

[2.4 Исследование зависимости погрешности 15](#_Toc182589094)

[2.5 Примеры работы приложения 24](#_Toc182589095)

[3 Вывод 25](#_Toc182589096)

[4 Список литературы 26](#_Toc182589097)

[5 Приложение 27](#_Toc182589098)

[5.1 Многослойный персептрон 27](#_Toc182589099)

[5.2 Обучающая выборка 29](#_Toc182589100)

# Цель работы

Целью работы является разработка мобильного приложения, которое сможет использовать нейросеть.

# Практическая часть

## Выбор приложения

В ходе лабораторной работы нам необходимо реализовать мобильное приложение с нейросетью: воспользуемся многослойной нейронной сетью, реализованный в ходе курса предмета «Нейроинформатика»[[1]](#footnote-1).

## Выбор предметной области и подготовка обучающей выборки

Для начала нам необходимо выбрать предметную область. Пусть это будет классификация видов апельсинов. Для отбора числовых характеристик и составления их диапазонов воспользуемся данными из [2]. В итоге получим следующую таблицу (рисунок 2.1).

Изображение выглядит как текст, снимок экрана, линия, число

Автоматически созданное описание

Рисунок 2.1 – Диапазоны характеристик разных видов апельсинов

Теперь нужно сгенерировать обучающую выборку, для чего воспользуемся сайтом [1]. На рисунках 2.2 и 2.3 приведен пример создания выборки для апельсина вида «Ambiance».

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 2.2 – Задание диапазонов характеристик «Ambiance»

Изображение выглядит как одежда, ткань, шаблон

Автоматически созданное описание

Рисунок 2.3 – Полученная выборка «Ambiance»

В итоге после приведения сгенерированных данных к виду, с которым сможет работать программа, мы получим обучающую выборку, изображенную на рисунке 2.4.

Изображение выглядит как снимок экрана, шаблон

Автоматически созданное описание

Рисунок 2.4 – Обучающая выборка[[2]](#footnote-2)

## Реализация работы многослойного персептрона

Сначала необходимо подключить библиотеки и модули, которые понадобятся для написания программы (рисунок 2.5):

* модуль «random» – предоставляет функции для генерации случайных чисел, букв, случайного выбора элементов последовательности;
* модуль «matplotlib.pyplot» – помогает автоматически создавать оси, фигуры и другие компоненты;
* библиотека «NumPy» – предоставляет реализации вычислительных алгоритмов (в виде функций и операторов), оптимизированные для работы с многомерными массивами;
* выборка «dataset» – созданная нами обучающая выборка.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 2.5 – Подключение библиотек и модулей

Теперь зададим количество нейронов на разных слоях нейронной сети, как показано на рисунке 2.6:

* во входном слое «input\_layer» будет 10 нейронов, так как у каждого апельсина 10 числовых характеристик;
* в выходном слое «output\_layer» – 4 нейрона, так как столько видов апельсинов;
* в скрытом слое «first\_layer» будет 100 нейронов – случайный выбор, который при необходимости будет изменён.

Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание

Рисунок 2.6 – Количество нейронов в разных слоях

Далее с помощью функции «np.random.rand(размерность полученной случайной матрицы)» инициализируем веса и векторы смещения для первого и второго слоя нейронной сети, как приведено на рисунке 2.7.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2.7 – Инициализация весов и векторов смещения

Создадим следующие гиперпараметры (рисунок 2.8):

* скорость обучения – «ALPHA»,
* количество эпох – «NUM\_EPOCHS»,
* размер батча – «BATCH\_SIZE»,

их значения определяются случайно и меняются в случае необходимости для повышения точности работы нейросети;

* массив для хранения вычисленной ошибки после каждой итерации – «loss\_arr».

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 2.8 – Задание гиперпараметров

Запустим цикл для обучения нейронной сети. Для начала создадим цикл «for», который будет выполняться «NUM\_EPOCHS» раз. Затем с помощью функции «random.shuffle» случайно перемешаем обучающую выборку, и запустим второй цикл «for», как показано на рисунке 2.9.

Изображение выглядит как текст, Шрифт, программное обеспечение, снимок экрана

Автоматически созданное описание

Рисунок 2.9 – Запуск обучения нейронной сети

Теперь напишем прямое распространение: начальные значения умножаем на веса первого слоя и изменяем на вектор смещения, после чего активируем полученные значения функцией «relu»[[3]](#footnote-3), затем все повторяем, но полученные веса передаём в функции активации «softmax\_batch». После этого вычисляем ошибку между полученными и фактическими значениями с помощью «sparse\_cross\_entropy\_batch» и сохраняем её в «loss\_arr» (рисунок 2.10).

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рисунок 2.10 – Прямое распространение

Далее, пользуясь правилами работы метода обратного распространения ошибки, реализуем её с помощью возможностей, предоставляемых библиотекой «NumPy», как изображено на рисунке 2.11.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2.11 – Обратное распространение

После этого необходимо обновить веса, для чего из старого значения вычитаем вычисленное в обратном распространении, которое нужно умножить на скорость обучения (рисунок 2.12). Далее начинается следующая итерация цикла «for».

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 2.12 – Обновление весов

После выполнения цикла и завершения обучения нейронной сети функция «calc\_accurcacy» вычисляет точность, которая при помощи «print» будет выведена в консоль. Затем «plt.plot» постоит, а «plt.show» отобразит график падения ошибки по итерациям (рисунок 2.13).

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 2.13 – Вычисление ошибки и построение графика падения

В конце запустим бесконечный цикл «while True», в котором пользователь сможет вводить 10 характеристик. По ним обученная нейронная сеть будет определять, к какому виду относится апельсин с введенными характеристиками, и выводить в консоль полученный ответ (рисунок 2.14).

Изображение выглядит как текст, программное обеспечение, Шрифт, снимок экрана

Автоматически созданное описание

Рисунок 2.14 – Использование обученной нейронной сети

Ниже представлены функции, которые были использованы ранее.

Функция «relu» возвращает максимальное значение из 0 и «t», которое передаётся в функцию (рисунок 2.15).

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Рисунок 2.15 – Функция активации «relu»

Функция «relu\_deriv» – это производная функции активации «relu». Функция «relu» принимает значения 0 или «t»: если 0, то «relu\_deriv» вернёт 0, а если «t», то – 1. Поэтому реализация «relu\_deriv» следующая: сначала переданное значение сравнивается с 0, после чего получается булева переменная, которая переводится в тип данных «float», как показано на рисунке 2.16.

Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание

Рисунок 2.16 – Функция «relu\_deriv»

Функции «softmax» и «softmax\_batch» работают аналогично: сначала поэлементно считается экспонента, которая затем делится на сумму по всем элементам (рисунок 2.17).

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рисунок 2.17 – Функции «softmax» и «softmax\_batch»

Функция «sparse\_cross\_entropy\_batch»: высчитываем кросс-энтропию для каждого вектора-строки из «z» и соответствующего правильного индекса из «y» (Рисунок 2.18).

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2.18 – Функция «sparse\_cross\_entropy\_batch»

Функция «to\_full\_batch»: сначала с помощью «np.zeros» создаем матрицу, которая будет заполнена нулями. Затем циклом «for» расставим единицы в соответствующих позициях, как приведено на рисунке 2.19.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 2.19 – Функция «to\_full\_batch»

Функция «predict»: работает так же, как и прямое распространение, которое уже было описано выше (рисунок 2.20).

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 2.20 – Функция «predict»

Функция «calc\_accuracy» вычисляет точность. Установим счетчик «correct» и запустим цикл «for»: если нейросеть выдаст правильный ответ, то счетчик увеличится на 1. После цикла разделим количество правильных ответов на количество примеров в обучающей выборке.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 2.21 – Функция «calc\_accuracy»

## Исследование зависимости погрешности

Обучим нейросеть с разными объёмами обучающей выборки и посмотрим на точность, которая получится (рисунок 2.22 и рисунок 2.23).

Изображение выглядит как текст, снимок экрана, дисплей, График

Автоматически созданное описание

Рисунок 2.22 – Точность при объеме выборки = 120 (по 30 примеров на каждый вид апельсина)

Изображение выглядит как текст, снимок экрана, дисплей, График

Автоматически созданное описание

Рисунок 2.23 – Точность при объеме выборки = 40 (по 10 примеров на каждый вид апельсина)

Теперь обучим нейронную сеть с разным количеством нейронов, как показано на рисунках 2.24 и 2.25.

Изображение выглядит как текст, дисплей, снимок экрана, Прямоугольник

Автоматически созданное описание

Рисунок 2.24 – Точность при 10 нейронах на скрытом слое

Изображение выглядит как текст, снимок экрана, дисплей, График

Автоматически созданное описание

Рисунок 2.25 – Точность при 100 нейронах на скрытом слое

Далее запустим обучения с разными размерами батча (рисунок 2.26 и рисунок 2.27).

Изображение выглядит как текст, снимок экрана, дисплей, График

Автоматически созданное описание

Рисунок 2.26 – Точность при размере батча = 50

Изображение выглядит как текст, снимок экрана, График

Автоматически созданное описание

Рисунок 2.27 – Точность при размере батча = 20

Наконец сравним точность при разных количествах итераций, как приведено на рисунках 2.28 и 2.29.

Изображение выглядит как текст, снимок экрана, дисплей, График

Автоматически созданное описание

Рисунок 2.28 – Точность при 1000 эпохах

Изображение выглядит как текст, снимок экрана, Шрифт, График

Автоматически созданное описание

Рисунок 2.29 – Точность при 100 эпохах

Проведя исследование зависимостей можно установить, что для решения задачи классификации апельсинов лучше использовать следующие параметры:

* объем выборки – 120 примеров характеристик разных видов апельсинов (по 30 примеров на каждый);
* количество нейронов на скрытом слое – 100;
* размер батча – 50;
* количество эпох – 1000.

## Примеры работы приложения

Для запуска приложения на «Android» воспользуемся «Python CodePad» [3]. На рисунке 2.30 и рисунке 2.31 представлена работа созданного мобильного приложения.

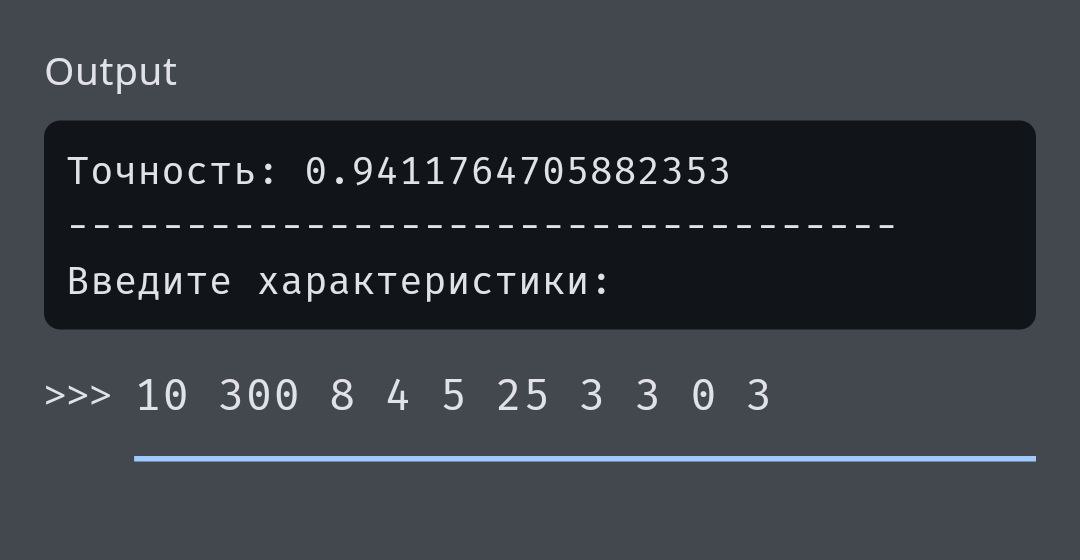


Рисунок 2.30 – Ввод параметров

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2.31 – Результат

# Вывод

В ходе лабораторной работы мы разработали мобильное приложение, которое использует нейросеть.

# Список литературы

1. https://www.mockaroo.com/
2. https://www.kaggle.com/datasets/shruthiiiee/orange-quality/data
3. https://play.google.com/store/apps/details?id=com.markodevcic.python\_code\_pad

# Приложение

## Многослойный персептрон

import random  
import numpy as np  
import matplotlib.pyplot as plt  
from dataset import dataset  
  
  
def relu(t):  
 return np.maximum(t**, 0**)  
  
  
def relu\_deriv(t):  
 return (t >= **0**).astype(float)  
  
  
def softmax(t):  
 out = np.exp(t)  
 return out / np.sum(out)  
  
  
def softmax\_batch(t):  
 out = np.exp(t)  
 return out / np.sum(out**,** axis=**1,** keepdims=True)  
  
  
def sparse\_cross\_entropy\_batch(z**,** y):  
 return -np.log(np.array([z[j**,** y[j]] for j in range(len(y))]))  
  
  
def to\_full\_batch(y**,** num\_classes):  
 y\_full = np.zeros((len(y)**,** num\_classes))  
 for j**,** yj in enumerate(y):  
 y\_full[j**,** yj] = **1** return y\_full  
  
  
def predict(x):  
 t1 = x @ W1 + b1  
 h1 = relu(t1)  
 t2 = h1 @ W2 + b2  
 z = softmax(t2)  
  
 return z  
  
  
def calc\_accuracy():  
 correct = **0** for x**,** y in dataset:  
 z = predict(x)  
 y\_pred = np.argmax(z)  
 if y\_pred == y:  
 correct += **1** acc = correct / len(dataset)  
 return acc  
  
  
input\_layer = **10**output\_layer = **4**first\_layer = **100**W1 = np.random.rand(input\_layer**,** first\_layer)  
b1 = np.random.rand(**1,** first\_layer)  
W2 = np.random.rand(first\_layer**,** output\_layer)  
b2 = np.random.rand(**1,** output\_layer)  
  
W1 = (W1 - **0.5**) \* **2** \* np.sqrt(**1**/input\_layer)  
b1 = (b1 - **0.5**) \* **2** \* np.sqrt(**1**/input\_layer)  
W2 = (W2 - **0.5**) \* **2** \* np.sqrt(**1**/first\_layer)  
b2 = (b2 - **0.5**) \* **2** \* np.sqrt(**1**/first\_layer)  
  
ALPHA = **0.00001**NUM\_EPOCHS = **100000**BATCH\_SIZE = **50**loss\_arr = []  
  
for ep in range(NUM\_EPOCHS):  
 random.shuffle(dataset)  
 for i in range(len(dataset) // BATCH\_SIZE):  
  
 batch\_x**,** batch\_y = zip(\*dataset[i\*BATCH\_SIZE: i\*BATCH\_SIZE+BATCH\_SIZE])  
 x = np.concatenate(batch\_x**,** axis=**0**)  
 y = np.array(batch\_y)  
  
 # Forward  
  
 t1 = x @ W1 + b1  
 h1 = relu(t1)  
 t2 = h1 @ W2 + b2  
 z = softmax\_batch(t2)  
  
 E = np.sum(sparse\_cross\_entropy\_batch(z**,** y))  
 loss\_arr.append(E)  
  
 # Backward  
  
 y\_full = to\_full\_batch(y**,** output\_layer)  
 dE\_dt2 = z - y\_full  
 dE\_dW2 = h1.T @ dE\_dt2  
 dE\_db2 = np.sum(dE\_dt2**,** axis=**0,** keepdims=True)  
 dE\_dh1 = dE\_dt2 @ W2.T  
 dE\_dt1 = dE\_dh1 \* relu\_deriv(t1)  
 dE\_dW1 = x.T @ dE\_dt1  
 dE\_db1 = np.sum(dE\_dt1**,** axis=**0,** keepdims=True)  
  
 # Update  
  
 W1 = W1 - ALPHA \* dE\_dW1  
 b1 = b1 - ALPHA \* dE\_db1  
 W2 = W2 - ALPHA \* dE\_dW2  
 b2 = b2 - ALPHA \* dE\_db2  
  
accuracy = calc\_accuracy()  
print("Точность:"**,** accuracy)  
  
plt.plot(loss\_arr)  
plt.show()  
  
while True:  
 print("-" \* **200**)  
 a1**,** a2**,** a3**,** a4**,** a5**,** a6**,** a7**,** a8**,** a9**,** a10 = map(float**,** input("Введите характеристики: ").split())  
 x = np.array([a1**,** a2**,** a3**,** a4**,** a5**,** a6**,** a7**,** a8**,** a9**,** a10])  
 probs = predict(x)  
 pred\_class = np.argmax(probs)  
 class\_names = ['Ambiance'**,** 'Cara Cara'**,** 'Hamlin'**,** 'Blood Orange']  
 print('Полученный класс:'**,** class\_names[pred\_class])

## Обучающая выборка

dataset = [  
 ([[**7.5, 209, 13.2, 3.18, 4, 20, 2.4, 1, 1, 4.2**]]**, 0**)**,** ([[**8.1, 193, 14.0, 2.96, 4, 14, 3.0, 2, 1, 3.9**]]**, 0**)**,** ([[**8.4, 188, 15.4, 3.91, 2, 21, 3.2, 2, 1, 4.9**]]**, 0**)**,** ([[**7.9, 221, 15.1, 3.1, 1, 15, 2.1, 2, 1, 4.7**]]**, 0**)**,** ([[**7.7, 241, 14.7, 3.17, 2, 22, 3.0, 1, 1, 4.6**]]**, 0**)**,** ([[**7.2, 228, 15.2, 3.44, 1, 19, 3.2, 1, 1, 3.7**]]**, 0**)**,** ([[**8.3, 187, 13.6, 3.68, 2, 23, 3.4, 2, 1, 3.9**]]**, 0**)**,** ([[**8.5, 221, 14.2, 3.92, 4, 20, 2.4, 1, 1, 4.3**]]**, 0**)**,** ([[**7.8, 179, 15.0, 3.93, 4, 23, 2.3, 2, 1, 4.7**]]**, 0**)**,** ([[**7.8, 201, 15.4, 3.19, 2, 23, 2.6, 2, 1, 4.1**]]**, 0**)**,** ([[**7.2, 196, 14.8, 3.23, 4, 17, 3.0, 1, 1, 5.0**]]**, 0**)**,** ([[**8.4, 258, 14.0, 3.88, 3, 17, 2.4, 2, 1, 4.5**]]**, 0**)**,** ([[**9.1, 204, 15.3, 3.55, 3, 21, 2.1, 1, 1, 4.9**]]**, 0**)**,** ([[**9.2, 205, 14.9, 3.4, 4, 18, 2.3, 1, 1, 4.4**]]**, 0**)**,** ([[**8.9, 174, 13.0, 3.08, 4, 19, 2.7, 2, 1, 3.9**]]**, 0**)**,** ([[**9.3, 204, 14.4, 3.87, 4, 22, 2.7, 2, 1, 4.3**]]**, 0**)**,** ([[**7.3, 243, 15.7, 3.45, 3, 18, 2.7, 2, 1, 4.4**]]**, 0**)**,** ([[**8.8, 181, 14.4, 3.79, 2, 23, 2.3, 1, 1, 3.6**]]**, 0**)**,** ([[**9.2, 228, 14.5, 3.4, 4, 23, 2.4, 2, 1, 4.1**]]**, 0**)**,** ([[**9.2, 220, 13.4, 3.23, 3, 16, 2.4, 1, 1, 4.6**]]**, 0**)**,** ([[**8.6, 240, 14.6, 3.8, 3, 22, 2.7, 1, 1, 3.7**]]**, 0**)**,** ([[**8.7, 199, 15.9, 3.03, 4, 18, 2.8, 2, 1, 4.7**]]**, 0**)**,** ([[**8.7, 246, 13.3, 3.08, 4, 22, 2.6, 2, 1, 4.3**]]**, 0**)**,** ([[**7.9, 190, 13.3, 3.18, 4, 16, 2.5, 1, 1, 4.5**]]**, 0**)**,** ([[**7.3, 201, 15.6, 3.43, 1, 23, 2.7, 2, 1, 4.5**]]**, 0**)**,** ([[**8.3, 235, 16.0, 3.12, 1, 23, 3.4, 2, 1, 3.8**]]**, 0**)**,** ([[**9.4, 260, 14.1, 2.89, 1, 20, 3.1, 2, 1, 4.2**]]**, 0**)**,** ([[**7.3, 220, 14.3, 3.14, 4, 15, 2.4, 1, 1, 3.8**]]**, 0**)**,** ([[**7.4, 252, 13.2, 3.93, 3, 18, 3.5, 2, 1, 3.7**]]**, 0**)**,** ([[**8.3, 258, 15.7, 3.92, 3, 19, 2.5, 1, 1, 4.0**]]**, 0**)**,** ([[**8.9, 174, 9.5, 4.12, 4, 10, 3.3, 4, 1, 2.9**]]**, 1**)**,** ([[**6.3, 239, 12.5, 3.98, 5, 13, 1.3, 4, 1, 2.6**]]**, 1**)**,** ([[**6.7, 215, 12.2, 4.01, 5, 9, 3.2, 4, 1, 2.6**]]**, 1**)**,** ([[**9.0, 286, 7.4, 3.96, 5, 11, 3.7, 2, 1, 3.1**]]**, 1**)**,** ([[**8.4, 253, 12.5, 4.32, 3, 12, 1.5, 4, 1, 2.7**]]**, 1**)**,** ([[**7.9, 178, 6.2, 3.88, 5, 16, 4.1, 2, 1, 2.7**]]**, 1**)**,** ([[**8.5, 220, 6.9, 3.87, 4, 13, 2.8, 2, 1, 2.9**]]**, 1**)**,** ([[**7.9, 228, 6.1, 3.27, 4, 7, 2.5, 3, 1, 3.1**]]**, 1**)**,** ([[**7.2, 236, 10.8, 3.19, 4, 8, 2.7, 2, 1, 2.7**]]**, 1**)**,** ([[**9.8, 141, 7.9, 3.57, 4, 9, 1.9, 4, 1, 2.6**]]**, 1**)**,** ([[**8.0, 135, 8.2, 4.04, 3, 15, 2.8, 4, 1, 3.1**]]**, 1**)**,** ([[**7.0, 157, 6.7, 4.07, 3, 16, 3.8, 3, 1, 3.0**]]**, 1**)**,** ([[**8.0, 183, 8.2, 3.41, 4, 14, 4.0, 3, 1, 2.9**]]**, 1**)**,** ([[**6.5, 124, 7.6, 3.85, 5, 14, 1.1, 3, 1, 2.9**]]**, 1**)**,** ([[**6.9, 113, 12.5, 3.93, 3, 16, 4.9, 2, 1, 2.8**]]**, 1**)**,** ([[**6.3, 262, 12.6, 3.53, 3, 11, 3.7, 2, 1, 3.2**]]**, 1**)**,** ([[**7.9, 263, 9.4, 3.39, 5, 9, 1.4, 4, 1, 3.5**]]**, 1**)**,** ([[**7.7, 285, 11.3, 4.18, 5, 10, 3.8, 3, 1, 3.2**]]**, 1**)**,** ([[**7.9, 238, 11.4, 3.28, 4, 8, 4.4, 2, 1, 3.4**]]**, 1**)**,** ([[**6.7, 145, 11.6, 3.18, 4, 12, 1.0, 3, 1, 3.2**]]**, 1**)**,** ([[**7.1, 257, 9.6, 3.64, 5, 13, 1.3, 3, 1, 3.3**]]**, 1**)**,** ([[**9.8, 250, 12.1, 3.65, 5, 9, 2.5, 4, 1, 3.0**]]**, 1**)**,** ([[**8.1, 291, 6.9, 3.85, 4, 8, 3.1, 3, 1, 2.9**]]**, 1**)**,** ([[**9.3, 195, 12.2, 4.01, 5, 9, 3.9, 2, 1, 2.8**]]**, 1**)**,** ([[**9.0, 180, 8.6, 3.51, 3, 8, 4.2, 3, 1, 2.9**]]**, 1**)**,** ([[**8.0, 243, 6.7, 3.54, 5, 9, 4.0, 3, 1, 3.3**]]**, 1**)**,** ([[**9.8, 189, 6.7, 4.38, 3, 16, 4.6, 2, 1, 2.6**]]**, 1**)**,** ([[**8.0, 127, 9.9, 3.38, 5, 15, 2.0, 3, 1, 3.3**]]**, 1**)**,** ([[**6.2, 239, 11.1, 3.84, 4, 14, 4.3, 4, 1, 3.1**]]**, 1**)**,** ([[**7.9, 218, 10.5, 3.39, 3, 11, 4.0, 2, 0, 4.5**]]**, 2**)**,** ([[**7.9, 216, 11.5, 3.24, 3, 13, 4.4, 4, 0, 3.5**]]**, 2**)**,** ([[**8.3, 199, 10.9, 3.22, 3, 11, 4.3, 2, 0, 4.3**]]**, 2**)**,** ([[**8.0, 197, 10.8, 3.38, 3, 14, 4.1, 2, 0, 4.5**]]**, 2**)**,** ([[**7.8, 206, 11.2, 3.33, 3, 11, 4.3, 2, 0, 4.5**]]**, 2**)**,** ([[**8.0, 202, 10.8, 3.36, 3, 15, 4.1, 2, 0, 3.6**]]**, 2**)**,** ([[**8.0, 191, 10.9, 3.26, 3, 12, 4.1, 2, 0, 3.9**]]**, 2**)**,** ([[**8.0, 208, 10.5, 3.25, 3, 15, 4.4, 2, 0, 3.7**]]**, 2**)**,** ([[**8.3, 192, 10.7, 3.33, 3, 13, 4.2, 2, 0, 3.8**]]**, 2**)**,** ([[**8.3, 195, 11.4, 3.24, 3, 12, 4.5, 2, 0, 4.4**]]**, 2**)**,** ([[**8.1, 217, 10.7, 3.34, 3, 13, 4.4, 2, 0, 4.1**]]**, 2**)**,** ([[**8.0, 206, 11.0, 3.32, 3, 14, 4.5, 2, 0, 4.2**]]**, 2**)**,** ([[**8.0, 216, 10.7, 3.29, 3, 11, 4.0, 2, 0, 4.3**]]**, 2**)**,** ([[**8.0, 196, 11.4, 3.21, 3, 12, 4.5, 2, 0, 3.9**]]**, 2**)**,** ([[**7.9, 193, 11.4, 3.37, 3, 12, 4.3, 2, 0, 4.3**]]**, 2**)**,** ([[**7.8, 190, 11.4, 3.3, 3, 12, 4.3, 4, 0, 3.6**]]**, 2**)**,** ([[**8.2, 218, 11.4, 3.22, 3, 12, 4.4, 2, 0, 4.0**]]**, 2**)**,** ([[**7.9, 203, 10.7, 3.3, 3, 13, 4.3, 2, 0, 4.1**]]**, 2**)**,** ([[**7.9, 199, 11.4, 3.24, 3, 12, 4.3, 4, 0, 3.7**]]**, 2**)**,** ([[**7.8, 204, 10.9, 3.31, 3, 13, 4.4, 4, 0, 4.0**]]**, 2**)**,** ([[**7.8, 205, 11.2, 3.21, 3, 12, 4.1, 2, 0, 3.8**]]**, 2**)**,** ([[**8.2, 195, 10.8, 3.35, 3, 14, 4.3, 4, 0, 3.8**]]**, 2**)**,** ([[**8.1, 215, 10.7, 3.2, 3, 13, 4.0, 2, 0, 3.8**]]**, 2**)**,** ([[**7.9, 204, 10.6, 3.21, 3, 12, 4.4, 2, 0, 4.2**]]**, 2**)**,** ([[**8.3, 196, 11.1, 3.27, 3, 14, 4.4, 4, 0, 3.6**]]**, 2**)**,** ([[**7.9, 218, 11.0, 3.24, 3, 12, 4.2, 4, 0, 3.9**]]**, 2**)**,** ([[**8.3, 194, 10.9, 3.31, 3, 13, 4.0, 4, 0, 4.2**]]**, 2**)**,** ([[**7.8, 216, 10.9, 3.34, 3, 14, 4.2, 2, 0, 3.5**]]**, 2**)**,** ([[**8.2, 197, 11.3, 3.4, 3, 11, 4.2, 2, 0, 4.1**]]**, 2**)**,** ([[**8.2, 215, 10.6, 3.33, 3, 14, 4.2, 4, 0, 4.5**]]**, 2**)**,** ([[**9.1, 290, 7.9, 4.02, 4, 25, 3.4, 3, 0, 3.4**]]**, 3**)**,** ([[**9.2, 273, 7.5, 4.0, 4, 21, 3.2, 3, 0, 3.3**]]**, 3**)**,** ([[**9.5, 290, 7.6, 4.02, 5, 21, 3.4, 3, 0, 3.4**]]**, 3**)**,** ([[**9.8, 271, 7.2, 4.05, 4, 24, 3.3, 3, 0, 3.1**]]**, 3**)**,** ([[**9.1, 291, 7.9, 3.95, 4, 23, 3.3, 3, 0, 3.1**]]**, 3**)**,** ([[**9.8, 261, 8.5, 3.95, 4, 23, 3.3, 3, 0, 3.2**]]**, 3**)**,** ([[**9.3, 271, 7.6, 3.91, 4, 25, 3.1, 3, 0, 3.5**]]**, 3**)**,** ([[**9.2, 276, 7.3, 3.89, 5, 21, 3.1, 3, 0, 3.2**]]**, 3**)**,** ([[**9.6, 264, 7.6, 3.92, 4, 25, 3.0, 3, 0, 3.3**]]**, 3**)**,** ([[**9.5, 290, 7.3, 3.98, 5, 24, 3.4, 3, 0, 3.0**]]**, 3**)**,** ([[**9.4, 271, 8.3, 3.94, 5, 22, 3.4, 3, 0, 3.3**]]**, 3**)**,** ([[**9.3, 269, 7.1, 3.88, 4, 21, 3.4, 3, 0, 3.4**]]**, 3**)**,** ([[**9.7, 282, 7.0, 3.94, 5, 21, 3.2, 3, 0, 3.3**]]**, 3**)**,** ([[**9.2, 277, 7.8, 4.06, 5, 23, 3.1, 3, 0, 3.0**]]**, 3**)**,** ([[**9.1, 267, 8.0, 3.82, 4, 25, 3.1, 3, 0, 3.4**]]**, 3**)**,** ([[**9.6, 263, 7.2, 3.94, 5, 23, 3.2, 3, 0, 3.4**]]**, 3**)**,** ([[**9.8, 258, 7.4, 3.95, 5, 22, 3.0, 3, 0, 3.4**]]**, 3**)**,** ([[**9.4, 299, 8.4, 3.94, 5, 22, 3.4, 3, 0, 3.3**]]**, 3**)**,** ([[**9.4, 262, 8.5, 3.84, 4, 22, 3.3, 3, 0, 3.3**]]**, 3**)**,** ([[**9.6, 253, 8.4, 3.94, 5, 23, 3.2, 3, 0, 3.1**]]**, 3**)**,** ([[**9.7, 295, 7.6, 4.0, 4, 21, 3.3, 3, 0, 3.4**]]**, 3**)**,** ([[**9.2, 279, 8.0, 3.96, 4, 24, 3.4, 3, 0, 3.2**]]**, 3**)**,** ([[**9.1, 256, 7.7, 3.89, 4, 22, 3.5, 3, 0, 3.1**]]**, 3**)**,** ([[**9.6, 279, 7.7, 4.05, 4, 24, 3.0, 3, 0, 3.4**]]**, 3**)**,** ([[**9.0, 287, 7.7, 3.87, 5, 23, 3.3, 3, 0, 3.2**]]**, 3**)**,** ([[**9.6, 265, 7.7, 4.08, 5, 24, 3.3, 3, 0, 3.5**]]**, 3**)**,** ([[**9.0, 264, 8.4, 3.84, 4, 22, 3.3, 3, 0, 3.3**]]**, 3**)**,** ([[**9.2, 271, 8.3, 3.96, 5, 21, 3.1, 3, 0, 3.2**]]**, 3**)**,** ([[**9.1, 250, 8.4, 3.86, 4, 21, 3.0, 3, 0, 3.1**]]**, 3**)**,** ([[**9.3, 260, 7.1, 3.97, 5, 22, 3.3, 3, 0, 3.1**]]**, 3**)**,**]

1. В пунктах 2.2 – 2.4 приведена разработка многослойного персептрона. [↑](#footnote-ref-1)
2. * Полная обучающая выборка [Обучающая выборка].

   [↑](#footnote-ref-2)
3. Эта и все остальные функции, которые применяются в обучении, вычислении ошибки и использовании нейронной сети, описаны далее. [↑](#footnote-ref-3)