МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра Вычислительной техники

Машинное обучение.

Лабораторная работа №1

Практика 1-2

|  |  |
| --- | --- |
| Факультет: АВТФ  Группа: АПИМ-24  Выполнил:  Разуваев В. В. | Проверил:  Гаврилов А.В. |

**Содержание**

[Цель работы 3](#_Toc184590788)

[Задание 3](#_Toc184590789)

[Ход работы 3](#_Toc184590790)

[Задача 1 3](#_Toc184590791)

[Задача 2 5](#_Toc184590792)

[Заключение 10](#_Toc184590793)

# Цель работы

Освоить базовые основы языка Python, а также создать простую нейронную сеть с использованием библиотеки Keras.

# Задание

1. Написать функцию, принимающая целое положительное число. Функция должна вернуть строку вида “(n1\*\*p1)(n2\*\*p2)…(nk\*\*pk)” представляющая разложение числа на простые множители (если pi == 1, то выводить только ni).

2. Дополнить фрагмент кода моделью ИНС, которая способна провести бинарную классификацию по сгенерированным данным:

# Ход работы

## Задача 1

*Листинг 1 – Код для разложения числа на простые множители*

def prime\_factors(n: int) -> list:

'''

разлагает число на простые множители и возвращает список этих множителей.

'''

i = 2 # Начинаем с наименьшего простого числа

factors = [] # Список для хранения простых множителей

while i \* i <= n: # Пока i в квадрате меньше или равно n

if n % i: # Если n не делится на i

i += 1 # Увеличиваем i на 1

else: # Если n делится на i

n //= i # Делим n на i

factors.append(i) # Добавляем i в список множителей

if n > 1: # Если n больше 1, значит оно само является простым числом

factors.append(n) # Добавляем n в список множителей

return factors # Возвращаем список множителей

def format\_factors(factors: list) -> str:

'''

принимает список множителей и форматирует их в строку в указанном формате

'''

factor\_count = {} # Словарь для подсчета количества каждого множителя

for factor in factors: # Проходим по каждому множителю

if factor in factor\_count: # Если множитель уже в словаре

factor\_count[factor] += 1 # Увеличиваем его счетчик

else: # Если множителя нет в словаре

factor\_count[factor] = 1 # Добавляем его со счетчиком 1

result = [] # Список для хранения отформатированных множителей

for factor, count in factor\_count.items(): # Проходим по каждому множителю и его счетчику

if count == 1: # Если счетчик равен 1

result.append(f"({factor})") # Добавляем множитель в формате (factor)

else: # Если счетчик больше 1

result.append(f"({factor}\*\*{count})") # Добавляем множитель в формате (factor\*\*count)

return ''.join(result) # Объединяем все элементы списка в одну строку и возвращаем её

def factorize(n):

factors = prime\_factors(n)

return format\_factors(factors)

if \_\_name\_\_ == "\_\_main\_\_":

numbers = (5,86240, 1247972, 100, 4628453, 551, 144, 1234525386396, 85546)

for i in numbers:

print(f"num = {i}: {factorize(i)}")

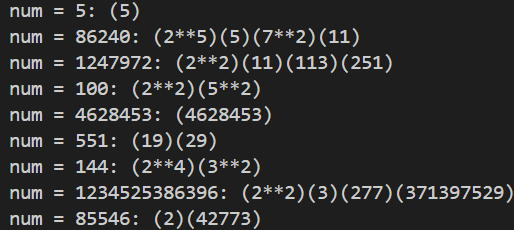


Рисунок 1 – Результат работы программы

## Задача 2

*Листинг 2 – Код для бинарной классификации*

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.colors as mclr

import math

from tensorflow.keras import layers

from tensorflow.keras import models

def genData(size=500):

size1 = size // 2

size2 = size - size1

t1 = np.random.rand(size1)

x1 = np.asarray([i \* math.cos(i \* 2 \* math.pi) + (np.random.rand(1) - 1) / 2 \* i for i in t1])

y1 = np.asarray([i \* math.sin(i \* 2 \* math.pi) + (np.random.rand(1) - 1) / 2 \* i for i in t1])

data1 = np.hstack((x1, y1))

label1 = np.zeros([size1, 1])

div1 = round(size1 \* 0.8)

t2 = np.random.rand(size2)

x2 = np.asarray([-i \* math.cos(i \* 2 \* math.pi) + (np.random.rand(1) - 1) / 2 \* i for i in t2])

y2 = np.asarray([-i \* math.sin(i \* 2 \* math.pi) + (np.random.rand(1) - 1) / 2 \* i for i in t2])

data2 = np.hstack((x2, y2))

label2 = np.ones([size2, 1])

div2 = round(size2 \* 0.8)

div = div1 + div2

order = np.random.permutation(div)

train\_data = np.vstack((data1[:div1], data2[:div2]))

test\_data = np.vstack((data1[div1:], data2[div2:]))

train\_label = np.vstack((label1[:div1], label2[:div2]))

test\_label = np.vstack((label1[div1:], label2[div2:]))

return (train\_data[order, :], train\_label[order, :]), (test\_data, test\_label)

def drawResults(data, label, prediction):

p\_label = np.array([round(x[0]) for x in prediction])

plt.scatter(data[:, 0], data[:, 1], s=30, c=label[:, 0], cmap=mclr.ListedColormap(['red', 'blue']))

plt.scatter(data[:, 0], data[:, 1], s=10, c=p\_label, cmap=mclr.ListedColormap(['red', 'blue']))

plt.grid()

plt.show()

# Генерация данных

(train\_data, train\_label), (test\_data, test\_label) = genData()

# Создание модели

model = models.Sequential()

model.add(layers.Dense(64, activation='relu', input\_shape=(train\_data.shape[1],)))

model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dense(1, activation='sigmoid'))

# Компиляция модели

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Обучение модели

H = model.fit(train\_data, train\_label, epochs=50, batch\_size=32, validation\_split=0.2)

# Получение ошибки и точности в процессе обучения

loss = H.history['loss']

val\_loss = H.history['val\_loss']

acc = H.history['accuracy']

val\_acc = H.history['val\_accuracy']

epochs = range(1, len(loss) + 1)

# Построение графика ошибки

plt.plot(epochs, loss, 'bo', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

# Построение графика точности

plt.clf()

plt.plot(epochs, acc, 'bo', label='Training acc')

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

plt.title('Training and validation accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

# Получение и вывод результатов на тестовом наборе

results = model.evaluate(test\_data, test\_label)

print(results)

# Вывод результатов бинарной классификации

all\_data = np.vstack((train\_data, test\_data))

all\_label = np.vstack((train\_label, test\_label))

pred = model.predict(all\_data)

drawResults(all\_data, all\_label, pred)

Функция genData генерирует два набора точек, принадлежащих к двум разным классам (0 и 1), распределённых по двумерным спиралям:

* Класс 0: Точки формируют спираль в положительном направлении.
* Класс 1: Точки формируют спираль в отрицательном направлении.
* Данные разделяются на обучающую и тестовую выборки в соотношении 80/20.

Создаётся простая нейронная сеть с использованием TensorFlow:

* Входной слой принимает два параметра (координаты точек: x и y).
* Два скрытых слоя с 64 нейронами, активация ReLU.
* Выходной слой с одним нейроном, активация sigmoid (для бинарной классификации).
* Функция потерь: binary\_crossentropy (логарифмическая функция для задач бинарной классификации).
* Оптимизатор: adam (адаптивная оптимизация градиентного спуска).
* Метрика: accuracy.
* Модель обучается на 50 эпохах с использованием мини-пакетов размером 32, при этом 20% обучающей выборки выделяется для валидации.

Строятся графики изменения ошибки (loss) и точности (accuracy) на обучающей и валидационной выборках по эпохам.

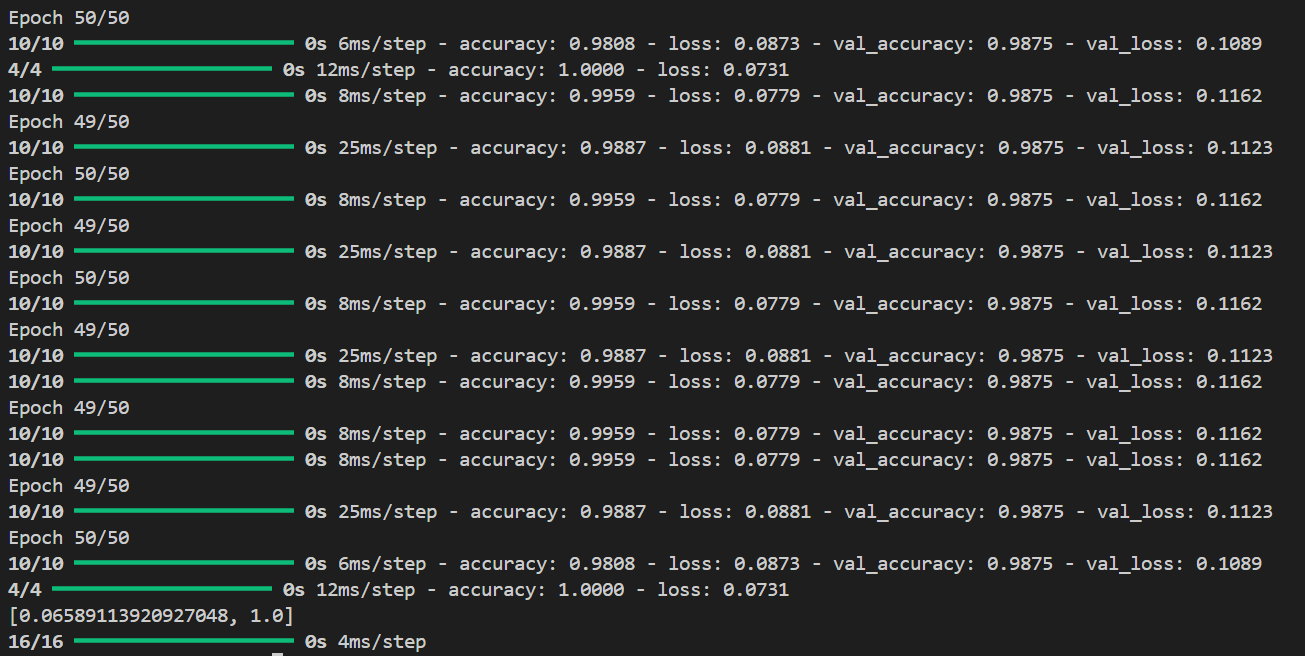


Рисунок 2 – Результат обучения нейронной сети

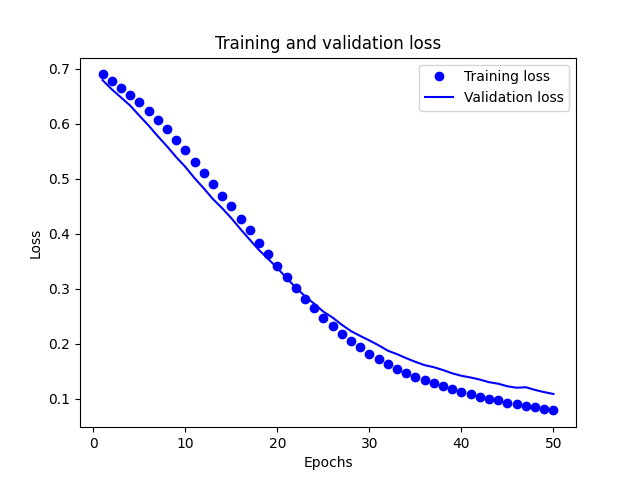


Рисунок 3 – График ошибки

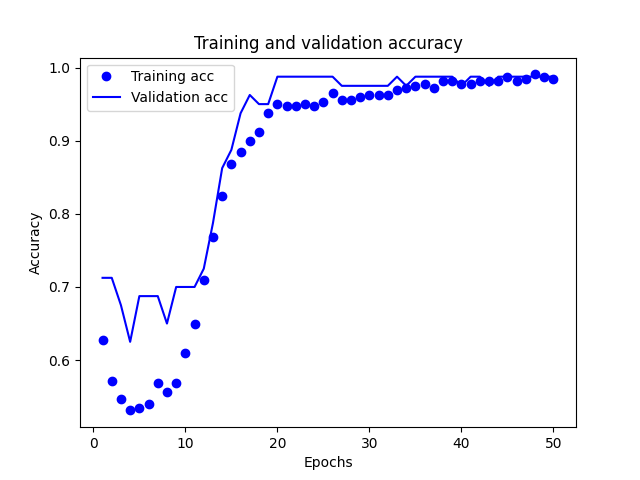


Рисунок 4 – График точности

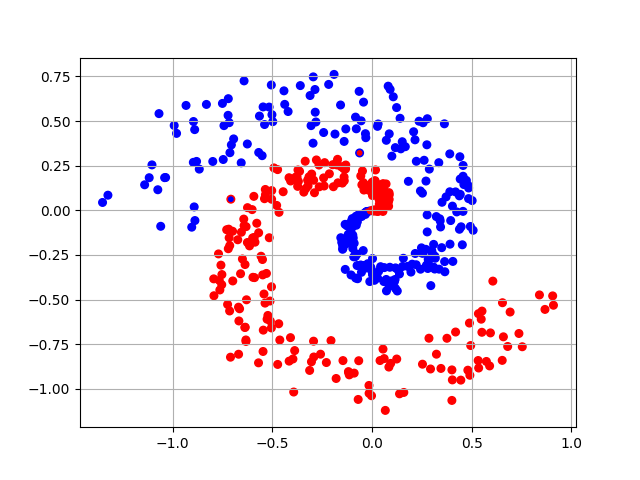


Рисунок 5 – Визуализация результатов классификации

# Заключение

В ходе выполнения лабораторной работы была написана пробная программа на Python, согласно индивидуальному варианту. Также была реализована моделью ИНС, способная провести бинарную классификацию по сгенерированным данным. Было произведено успешное обучение нейронной сети.