Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Кафедра прикладной математики и кибернетики

Отчет

по лабораторной работе №4

Выполнил:

студент гр. ИП-712

Алексеев С.В.

Проверил:

Ассистент кафедры  
Морозова К.И.

Новосибирск, 2020 г.

Оглавление

[Текст задания. 3](#_Toc59008733)

[Описание основных функций 3](#_Toc59008734)

[Результат работы программы 6](#_Toc59008735)

[Код программы 7](#_Toc59008736)

# Текст задания.

Целью данной лабораторной работы является разработка нейронной сети для решения задачи классификации или регрессии в зависимости от набора данных в рамках варианта. Лабораторная работа предполагает разработку на языке программирования Python с использованием библиотеки Keras.

Вариант задания:

3) Определение эмоционального окраса рецензии фильма (IMDB movie review sentiment classification dataset)

При разработке нейронной сети следует соблюсти наличие необходимых составляющих исходя из следующего варианта:

1) Нейросеть должна состоять из трёх полносвязных слоёв, обязательное использование Dropout, в качестве оптимизатора использовать Adam;

Выбор количества нейронов на всех внутренних слоях, функций активации и других параметров должен быть обусловлен оптимальностью работы модели.

Для защиты лабораторной работы следует обосновать выбор значений дополнительных параметров и продемонстрировать работу обученной нейронной сети. Обоснование должно включать в себя демонстрацию качества работы сети на валидационном наборе данных в процессе обучения. Параметры выбираются те, на которых валидация даёт наилучший результат.

# Описание основных функций

Keras-это библиотека Python с открытым исходным кодом для легкого построения нейронных сетей.

imdb.load\_data(num\_words=10000) - Загружает [набор данных IMDB.](https://ai.stanford.edu/~amaas/data/sentiment/) Это набор данных из 50 000 обзоров фильмов из IMDB, помеченных

(положительный / отрицательный). Рецензии были предварительно обработаны, и каждая рецензия является кодируется в виде списка индексов слов (целых чисел).

data = np.concatenate((training\_data, testing\_data), axis = 0) targets = np.concatenate((training\_targets, testing\_targets), axis = 0) Для избежания разделения test train 50/50, мы сразу же объединим данные в data и targets после загрузки, чтобы позже выполнить разделение. def vectorize(sequences, dimension = 10000): - векторизуем каждый отзыв и заполняем его нулями, чтобы он содержал ровно 10 000 чисел.

Заполняем каждый обзор, который короче 10 000, нулями.

data = vectorize(data)

targets = np.array(targets).astype("float32") test\_x = data[:10000] test\_y = targets[:10000] train\_x = data[10000:] train\_y = targets[10000:]

Разделили данные на train и test наборы. Train набор будет содержать

40.000 обзоров, а test набор - 10.000

model = models.Sequential() - Создание последовательной модели

model.add(layers.Dropout(0.2, input\_shape = (10000,)))

model.add(layers.Dense(50, activation = "relu", input\_shape = (10000,))) model.add(layers.Dropout(0.3, input\_shape = (50,))) model.add(layers.Dense(50, activation = "relu")) model.add(layers.Dropout(0.2, input\_shape = (50,))) model.add(layers.Dense(1, activation = "sigmoid")) model.summary()

model.compile(optimizer = "adam", loss = "binary\_crossentropy", metrics = ["accurac y"])

Далее добавили входные, скрытые и выходные слои. Между ними мы используем отсев, чтобы предотвратить переобучение. Коэффициент отсева равен от 0.2 до 0.5

model.compile(optimizer = "adam", loss = "binary\_crossentropy", metrics = ["accuracy"]) Используем оптимизатор “adam", алгоритм, который изменяет веса и смещения во время тренировки. Мы также Выбираем двоичную кроссэнтропию в качестве потери (потому что мы имеем дело с двоичной классификацией) и точность в качестве нашей оценочной метрики.

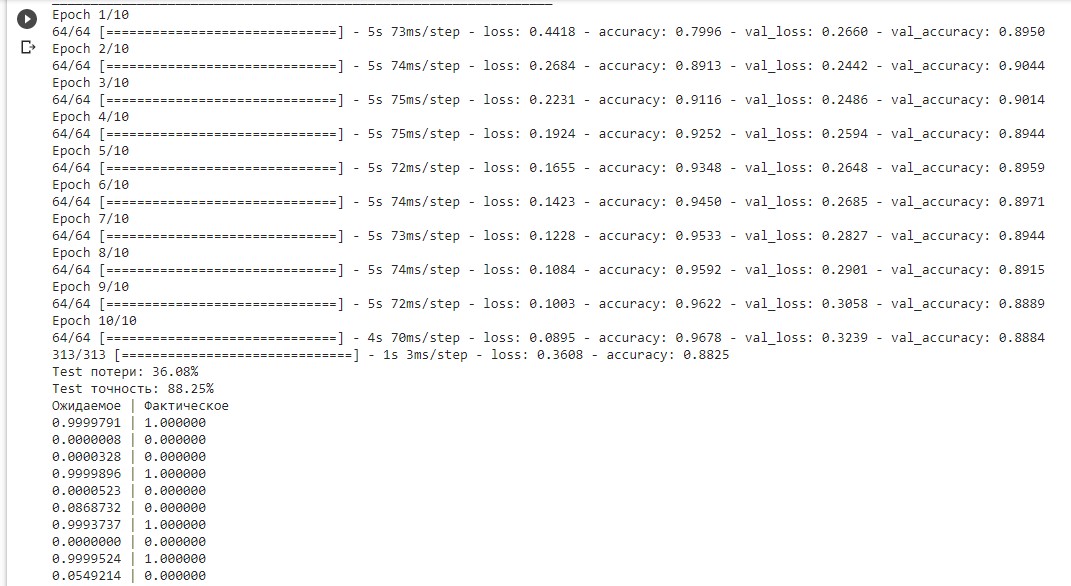
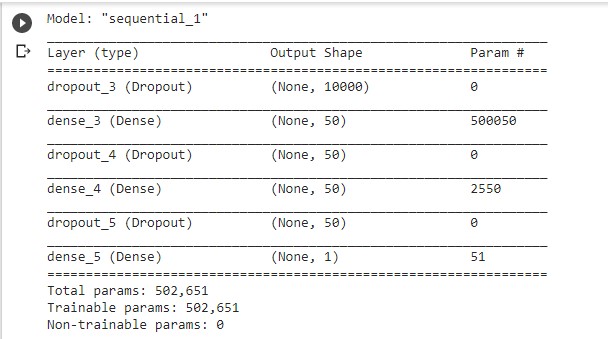
results = model.fit(train\_x, train\_y, epochs = 10, batch\_size = 500, validat ion\_split = 0.2) - Обучение сети

Тренируем нашу модель с размером пакета 500, для 10 «эпох»

Размер пакета определяет количество выборок, которые будут распространяться по сети, а эпоха - это итерация по всем обучающим данным score = model.evaluate(test\_x, test\_y) — возвращает значение потерь и значения метрик для модели в тестовом режиме

#Запуск распознавателя: prediction = model.predict(test\_x) — генерирует выходные прогнозы для входных выборок.

# Результат работы программы



# Код программы

import numpy as np

from keras import models from keras import layers from keras.datasets import imdb

(training\_data, training\_targets), (testing\_data, testing\_targets) = imdb.load\_data

(num\_words=10000) data = np.concatenate((training\_data, testing\_data), axis = 0) targets = np.concatenate((training\_targets, testing\_targets), axis = 0)

def vectorize(sequences, dimension = 10000):

results = np.zeros((len(sequences), dimension)) for i, sequence in enumerate(sequences):

results[i, sequence] = 1 return results

def toFixed(numObj, digits =0): return f"{numObj:.{digits}f}"

data = vectorize(data) targets = np.array(targets).astype("float32") test\_x = data[:10000] test\_y = targets[:10000] train\_x = data[10000:] train\_y = targets[10000:]

#Создание последовательной модели, добавление уровней сети и компиляция модели: model = models.Sequential()

model.add(layers.Dropout(0.2, input\_shape = (10000,))) model.add(layers.Dense(50, activation = "relu", input\_shape = (10000,)))

model.add(layers.Dropout(0.3, input\_shape = (50,))) model.add(layers.Dense(50, activation = "relu"))

model.add(layers.Dropout(0.2, input\_shape = (50,))) model.add(layers.Dense(1, activation = "sigmoid")) model.summary() model.compile(optimizer = "adam", loss = "binary\_crossentropy", metrics = ["accurac y"])

#Обучение сети:

results = model.fit(train\_x, train\_y, epochs = 10, batch\_size = 500, validation\_spl it = 0.2)

#Возвращает значение потерь и значения метрик для модели в тестовом режиме. score = model.evaluate(test\_x, test\_y) print("Test потери: %.2f%%" % (score[0] \* 100)) print("Test точность: %.2f%%" % (score[1] \* 100))

#Запуск распознавателя prediction = model.predict(test\_x) print("Ожидаемое | Фактическое") for i in range(10):

print('%.7f | %f' % (prediction[i], test\_y[i]))

]))