МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева (Самарский университет)»

Институт информатики, математики и электроники

Факультет информатики   
Кафедра программных систем

**ОТЧЕТ**

к лабораторной работе №2 по дисциплине

«Нейронные сети глубокого обучения»

Студент Е. Г. Плешаков

Студент О.В. Ширяева

Преподаватель А. Н. Жданова

Самара 2021

Содержание

[1 Постановка задачи 3](#_Toc66989919)

[2 Описание архитектуры сети, метода обучения 4](#_Toc66989920)

[3 Вычислительные эксперименты 8](#_Toc66989921)

[4 Результат работы программы на своих данных 8](#_Toc66989922)

[5 Вывод 9](#_Toc66989923)

[Приложение А Код программы 10](#_Toc66989924)

Постановка задачи

Цель лабораторной работы: реализовать нейронную сеть для прогнозирования временных рядов.

В общем виде задача прогнозирования временных рядов может быть сформулирована следующим образом. Пусть имеется некоторый источник, порождающий последовательность элементов *x1*, *x2*, . . . из некоторого множества *A*, называемого алфавитом. Алфавит может быть как конечным, так и бесконечным (т. е. представлять собой некоторый ограниченный непрерывный интервал). Пусть при этом на момент времени *t* мы имеем конечную порождённую источником последовательность *x1*, *x2*, *. . . , xt* . Задача прогнозирования сводится к предсказанию элемента, следующего в момент времени (*t+1*), т. е. элемента *xt+1*. Когда алфавит *A* является дискретным и конечным, любой алгоритм прогнозирования может быть применён к данному случаю естественным образом, так как будет оперировать с конечным множеством алфавита *A* и с конечной выборкой *x1*, *x2*, *. . . , xt*. Если алфавит *A* представляет собой непрерывный конечный интервал, то поступим следующим образом. Разобьём заданный интервал на фиксированное количество непересекающихся подмножеств (в общем случае подмножества могут быть произвольного неравного размера), сопоставим им целочисленные номера в соответствии с их порядком в исходном интервале. Количество возможных номеров будет совпадать с числом интервалов. При этом множество всех номеров будет представлять собой уже новый конечный дискретный алфавит *A*′. Далее преобразуем исходный временной ряд из терминов в алфавите *A* в ряд, записанный в терминах нового алфавита *A*′. Таким образом, получим некоторую конечную выборку (ряд) из конечного алфавита и будем работать с ним, как с конечным дискретным алфавитом. При этом после прогнозирования очередного значения такого ряда ему сопоставляются соответствующий его номеру непрерывный интервал или точка из него (например, центр интервала). Количество букв алфавита обозначим через *N*. Предполагается, что процесс, или источник информации, является стационарным и эргодическим, т. е. неформально распределение вероятностей символов этого источника не изменяется со временем и не зависит от конкретной реализации процесса. Пусть источник порождает сообщение *x1, . . . , xt−1, xt , xi ∈ A, i = 1, 2, . . . , t*, и требуется произвести прогнозирование n следующих элементов (в простейшем случае — одного элемента). Ошибкой прогноза называется (апостериорная) величина отклонения прогноза от действительного состояния объекта (т. е. величина |xi−xi∗|, где xi∗ — прогнозное значение, xi — реальное значение). Под ошибкой прогнозирования *n* элементов будем понимать среднюю ошибку прогноза каждого из *n* элементов в отдельности. Ошибка прогноза характеризует качество прогнозирования. Очевидно, если распределение вероятностей исходов процесса известно заранее, то задача прогнозирования следующих значений решается достаточно просто (строится прогнозная функция в соответствии с известной закономерностью либо прогнозируются значения исходя из удовлетворения плотности распределения вероятностей ряда, полученного после вставки прогнозных элементов). Однако в большинстве практических задач описанные априорные данные отсутствуют, да и не всегда заданное распределение явно существует. В настоящей лабораторной работе будет рассматриваться именно такой случай: прогнозирование заболеваемости covid-19.

1. Описание архитектуры сети, метода обучения

Для решения поставленной задачи была построена рекуррентная нейронная сеть, а точнее, её разновидность LSTM (Long short-term memory). Архитектура LSTM является наиболее подходящей для моделирования временных связей в глубоких нейронных сетях. Она преодолевает проблему исчезающего градиента в рекуррентной нейронной сети для долгосрочного обучения зависимости в данных с использованием ячеек памяти и вентилей.

LSTM-сеть — это искусственная нейронная сеть, содержащая LSTM-модули вместо или в дополнение к другим сетевым модулям. LSTM-модуль — это рекуррентный модуль сети, способный запоминать значения как на короткие, так и на длинные промежутки времени. Ключом к данной возможности является то, что LSTM-модуль не использует функцию активации внутри своих рекуррентных компонентов. Таким образом, хранимое значение не размывается во времени, и [градиент](https://ru.wikipedia.org/wiki/Градиент) или штраф не исчезает при использовании [метода обратного распространения ошибки во времени](https://ru.wikipedia.org/w/index.php?title=Метод_обратного_распространения_ошибки_во_времени&action=edit&redlink=1) при обучении искусственной нейронной сети.

LSTM-модули часто группируются в «блоки», содержащие различные LSTM-модули. Подобное устройство характерно для «глубоких» многослойных нейронных сетей и способствует выполнению параллельных вычислений с применением соответствующего оборудования. В формулах ниже каждая переменная, записанная строчным курсивом, обозначает вектор размерности равной числу LSTM-модулей в блоке.

LSTM-блоки содержат три или четыре «вентиля», которые используются для контроля потоков информации на входах и на выходах памяти данных блоков. Эти вентили реализованы в виде [логистической функции](https://ru.wikipedia.org/wiki/Логистическая_функция) для вычисления значения в диапазоне [0; 1]. Умножение на это значение используется для частичного допуска или запрещения потока информации внутрь и наружу памяти. Например, «входной вентиль» контролирует меру вхождения нового значения в память, а «вентиль забывания» контролирует меру сохранения значения в памяти. «Выходной вентиль» контролирует меру того, в какой степени значение, находящееся в памяти, используется при расчёте выходной функции активации для блока. (В некоторых реализациях входной вентиль и вентиль забывания воплощаются в виде единого вентиля. Идея заключается в том, что старое значение следует забывать тогда, когда появится новое значение, достойное запоминания).

Веса в LSTM-блоке (***W*** и ***U***) используются для задания направления оперирования вентилей. Эти веса определены для значений, которые подаются в блок (включая *xt* и выход с предыдущего временного шага *ht-1*) для каждого из вентилей. Таким образом, LSTM-блок определяет, как распоряжаться своей памятью как функцией этих значений, и тренировка весов позволяет LSTM-блоку выучить функцию, минимизирующую потери. LSTM-блоки обычно тренируют при помощи метода обратного распространения ошибки во времени.

Для обучения модели используются оптимизатор Adam и функция потерь MSEloss. Adam — один из самых эффективных алгоритмов оптимизации в обучении нейронных сетей. Он сочетает в себе идеи RMSProp и оптимизатора импульса. Вместо того чтобы адаптировать скорость обучения параметров на основе среднего первого момента (среднего значения), как в RMSProp, Adam также использует среднее значение вторых моментов градиентов. В частности, алгоритм вычисляет экспоненциальное скользящее среднее градиента и квадратичный градиент, а параметры beta1 и beta2 управляют скоростью затухания этих скользящих средних. MSELoss – простая функция потери, которая вычисляет среднеквадратичную ошибку между вводом и целью.

Далее приведём исходный код программы, снабженный комментариями.

1. Вычислительные эксперименты
2. Результат работы программы
3. Вывод

Приложение А  
Код программы

**Файл create\_cnn.py**

# Importing the Keras libraries and packages

from keras.layers import Conv2D

from keras.layers import Dense

from keras.layers import Flatten

from keras.layers import MaxPooling2D

from keras.models import Sequential

from tensorflow import keras

# Initialising the CNN

model = Sequential()

# Step 1 - Convolution

model.add(Conv2D(32, (3, 3), input\_shape=(128, 128, 3), activation='relu'))

# Step 2 - Pooling

model.add(MaxPooling2D(pool\_size=(2, 2)))

# Adding a second convolutional layer

# model.add(Conv2D(32, (3, 3), activation='relu'))

# model.add(MaxPooling2D(pool\_size=(2, 2)))

# Step 3 - Flattening

model.add(Flatten())

# Step 4 - Full connection

# model.add(Dense(units=256, activation='relu'))

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

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

# Compiling the CNN

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

# Part 2 - Fitting the CNN to the images

from keras.preprocessing.image import ImageDataGenerator

train\_datagen = ImageDataGenerator(rescale=1. / 255,

shear\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True)

test\_datagen = ImageDataGenerator(rescale=1. / 255)

batch\_size = 32

training\_set = train\_datagen.flow\_from\_directory('dataset/training\_set',

target\_size=(128, 128),

batch\_size=batch\_size,

class\_mode='binary')

test\_set = test\_datagen.flow\_from\_directory('dataset/test\_set',

target\_size=(128, 128),

batch\_size=batch\_size,

class\_mode='binary')

import os

cp\_callback = keras.callbacks.ModelCheckpoint(filepath=os.curdir,

save\_weights\_only=True,

verbose=1)

model.fit(training\_set,

steps\_per\_epoch=8000 // batch\_size,

epochs=25,

validation\_data=test\_set,

validation\_steps=2000 // batch\_size,

callbacks=[cp\_callback])

model.save("keras\_cnn")

print("Model saved")

**Файл create\_cnn.py**

import os

import matplotlib.pyplot as plt

import matplotlib

import numpy as np

import pandas as pd

import tensorflow as tf

from keras.preprocessing import image

from keras.preprocessing.image import ImageDataGenerator, load\_img

model = tf.keras.models.load\_model("keras\_cnn")

print("Model loaded")

train\_datagen = ImageDataGenerator(rescale=1. / 255,

shear\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True)

test\_datagen = ImageDataGenerator(rescale=1. / 255)

training\_set = train\_datagen.flow\_from\_directory('dataset/training\_set',

target\_size=(128, 128),

batch\_size=32,

class\_mode='binary')

test\_set = test\_datagen.flow\_from\_directory('dataset/test\_set',

target\_size=(128, 128),

batch\_size=32,

class\_mode='binary')

test\_filenames = os.listdir("dataset/control\_set")

test\_df = pd.DataFrame({

'filename': test\_filenames

})

nb\_samples = test\_df.shape[0]

sample\_test = test\_df.head(18)

sample\_test.head()

plt.figure(figsize=(6, 6))

for index, row in sample\_test.iterrows():

filename = row['filename']

img = load\_img("dataset/control\_set/" + filename, target\_size=(128, 128))

test\_image = image.load\_img("dataset/control\_set/" + filename, target\_size=(128, 128))

test\_image = image.img\_to\_array(test\_image)

test\_image = np.expand\_dims(test\_image, axis=0)

result = model.predict(test\_image)

if result[0][0] == 1:

prediction = 'dog'

else:

prediction = 'cat'

category = prediction

axes = plt.subplot(6, 4, index + 1)

plt.imshow(img)

plt.xlabel(format(category))

axes.set\_yticks([])

axes.set\_xticks([])

plt.tight\_layout()

plt.show()