МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №8

по дисциплине «Искусственные нейронные сети»

Тема: Генерация текста на основе “Алисы в стране чудес”

Студент гр. 7382 _____ Глазунов С.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2020

Цель работы.

Рекуррентные нейронные сети также могут быть использованы в качестве генеративных моделей.Это означает, что в дополнение к тому, что они используются для прогнозных моделей (создания прогнозов), они могут изучать последовательности проблемы, а затем генерировать совершенно новые вероятные последовательности для проблемной области.

Требования к выполнению задания.

1. Реализовать модель ИНС, которая будет генерировать текст.
2. Написать собственный CallBack, который будет показывать то как генерируется текст во время обучения (то есть раз в какое-то количество эпох генирировать и выводить текст у необученной модели).
3. Отследить процесс обучения при помощи TensorFlowCallBack, в отчете привести результаты и их анализ.

Ход работы.

1. Была построена и обучена модель нейронной сети, которая будет генерировать текст. Код предоставлен в приложении А.

Модель:

```
model = Sequential()
model.add(LSTM(256, input_shape=(X.shape[1], X.shape[2])))
model.add(Dropout(0.2))
model.add(Dense(y.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

2. Напишем собственный CallBack, который показывает, то как генерируется текст во время обучения.

```
def print_seq(model, epoch=0):
    start = numpy.random.randint(0, len(dataX) - 1)
    pattern = dataX[start]
    print("Seed:")

    print("\"", ".join([int_to_char[value] for value in pattern]), "\"")
    text = []
```

```
    for i in range(1000):
        x = numpy.reshape(pattern, (1, len(pattern), 1))
        x = x / float(n_vocab)
        prediction = model.predict(x, verbose=0)
        index = numpy.argmax(prediction)
        result = int_to_char[index]
        text.append(result)
        pattern.append(index)
        pattern = pattern[1:len(pattern)]
    with open('text_{}.txt'.format(epoch), 'w') as file:
        file.write(''.join(text))


class CB(callbacks.Callback):
    def __init__(self, epochs):
        super(CB, self).__init__()
        self.epochs = epochs

    def on_epoch_end(self, epoch, logs=None):
        if epoch in self.epochs:
            print_seq(self.model, epoch)
```

3. Отследим процесс обучения и рассмотрим тексты сгенерированные после [0, 5, 10, 15, 25] эпохи.

После 1 эпохи сеть сгенериловала повторяющуюся последовательность состоящую из 4 элементов.

oe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe
toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe
toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe
toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe
toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe
toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe
toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe
toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe
toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe
toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe t

После 10 эпохи сеть сгенериловала повторяющуюся последовательность но уже большей длины.

art to tee toree to the wooee
and the woote to the wooee to the tooee to the sooee to

the sooee to the

После 15 эпохи сеть сгенериловала текст, в нем можно рассмотреть настоящие слова и много непонятных.

the was oo the whil of the woole and the was oo the woile th the whele and the woile the wait on the woole and the woile tas ao the sooee of the whele and the woile tas ao the woile th the whele and the woile tas ao the woile th the whele and the woile the wait on the woole and the woile tas ao the sooee of the whele and the woile tas ao the woile th the whele and the woile tas ao the woile th the whele and the woile the wait on the woole and the woile tas ao the sooee of the whele and the woile tas ao the woile th the whele and the woile tas ao the woile th the whele and the woile the wait on the woole and the woile tas ao the sooee of the whele and the woile tas ao the woile th the whele and the woile tas ao the woile th the whele and the woile the wait on the woole and the woile tas ao the sooee of the whele and the woile tas ao the woile th the whele and the woile tas ao the woile th the whele and the woile the wait on the woole and the woile tas ao the sooe

После 25:

and ths soe thet sas to toe thet soe of the whrle haad oo the tas of the was

После 20 эпохи сеть уже начала генерировать разборчивый текст, с адекватными словами. Обучать сеть на большем количестве эпох не предоставляется возможном, т.к. сеть с 30 эпохами обучалась ~ 3-4 часа..

" ill she fancied
she heard the rabbit just under the window, she suddenly spread out her
hand, and ma "

Выводы.

Была построена и обучена нейронная сеть для генерации текстов на основе «Алисы в стране чудес». Был написан CallBack, с помощью которого отслеживался прогресс нейронной сети. В результате обучения сеть

научилась генерировать неосмысленные тексты, в которых встречаются слова.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
import sys
from os import listdir, path

import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import LSTM
from keras.callbacks import ModelCheckpoint, callbacks
from keras.utils import np_utils


filename = "wonderland.txt"
raw_text = open(filename).read()
raw_text = raw_text.lower()

chars = sorted(list(set(raw_text)))
char_to_int = dict((c, i) for i, c in enumerate(chars))
int_to_char = dict((i, c) for i, c in enumerate(chars))

n_chars = len(raw_text)
n_vocab = len(chars)

print("Total Characters: ", n_chars)
print("Total Vocab: ", n_vocab)

seq_length = 100
dataX = []
dataY = []

for i in range(0, n_chars - seq_length, 1):
    seq_in = raw_text[i:i + seq_length]
    seq_out = raw_text[i + seq_length]
    dataX.append([char_to_int[char] for char in seq_in])
    dataY.append(char_to_int[seq_out])


def print_seq(model, epoch=0):
    start = numpy.random.randint(0, len(dataX) - 1)
    pattern = dataX[start]
    print("Seed:")

    print("\"", ".join([int_to_char[value] for value in pattern]), "\"")
    text = []

    for i in range(1000):
        x = numpy.reshape(pattern, (1, len(pattern), 1))
        x = x / float(n_vocab)
```

```
        prediction = model.predict(x, verbose=0)
        index = numpy.argmax(prediction)
        result = int_to_char[index]
        text.append(result)
        pattern.append(index)
        pattern = pattern[1:len(pattern)]
    with open('text_{}.txt'.format(epoch), 'w') as file:
        file.write(''.join(text))


class CB(callbacks.Callback):
    def __init__(self, epochs):
        super(CB, self).__init__()
        self.epochs = epochs

    def on_epoch_end(self, epoch, logs=None):
        if epoch in self.epochs:
            print_seq(self.model, epoch)


n_patterns = len(dataX)
print("Total Patterns: ", n_patterns)

# reshape X to be [samples, time steps, features]
X = numpy.reshape(dataX, (n_patterns, seq_length, 1))
# normalize
X = X / float(n_vocab)
# one hot encode the output variable
y = np_utils.to_categorical(dataY)

model = Sequential()
model.add(LSTM(256, input_shape=(X.shape[1], X.shape[2])))
model.add(Dropout(0.2))
model.add(Dense(y.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')

filepath="weights-improvement-{epoch:02d}-{loss:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=1, save_best_only=True,
mode='min')
callbacks_list = [checkpoint, CB([0, 5, 10, 15, 25])]

model.fit(X, y, epochs=30, batch_size=128, callbacks=callbacks_list)

## generate ##

# choose filename
folder = '.'
filename = ''
min = 100000
for name in listdir(folder):
    full_name = path.join(folder, name)
    if path.isfile(full_name) and full_name.find('.hdf5') != -1:
```

```
        model_loss = int(full_name.split('.')[2])
        if min > model_loss:
            min = model_loss
            filename = full_name

print(filename)
model.load_weights(filename)
model.compile(loss='categorical_crossentropy', optimizer='adam')
print('FULL MODEL')
print_seq(model)
print("\nDone.")
```