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

ОТЧЕТ

по лабораторной работе №8

по дисциплине «Искусственные нейронные сети»

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

Студент гр. 8383 _____ Киреев К.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

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.

Подобные генеративные модели полезны не только для изучения того, насколько хорошо модель выявила проблему, но и для того, чтобы узнать больше о самой проблемной области.

Задачи

- Ознакомиться с генерацией текста
- Ознакомиться с системой Callback в Keras

Требования

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

Ход работы

Модель ИНС для генерации текста

Была реализована модель ИНС для генерации текста на основе “Алисы в стране чудес”. Листинг модели представлен ниже.

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

Собственный CallBack

Был написан собственный CallBack – GenerationCallback(), который раз в 5 эпох генерирует и выводит текст у необученной модели. Листинг представлен ниже.

```
class GenerationCallback(Callback):
    def gen_text(self, size=100):
        start = np.random.randint(0, n_patterns-1)
        pattern = dataX[start]
        text = []
        for i in range(size):
            x = np.reshape(pattern, (1, len(pattern), 1))
            x = x / float(n_vocab)
            prediction = model.predict(x, verbose=0)
            index = np.argmax(prediction)
            result = int_to_char[index]
            text.append(result)
            pattern.append(index)
            pattern = pattern[1:len(pattern)]
        print("".join(text))

    def on_epoch_end(self, epoch, logs=None):
        if epoch % 5 == 0:
            print(f'{epoch+1} ep:')
            self.gen_text()
```

Пример генерации текста на каждой пятой эпохе.

```
1 ep:
  aa the  aa the  aa the  aa the  aa the  aa the  aa the  aa the  aa the  aa the  aa the  aa the  aa

6 ep:
  e a lott oo the toeee to the toeee  and the soie to tee toeee ta the torel th the toree an the cadt

11 ep:
  the was anl noek the was anl nhek the was anl nhek the was anl nhek the was anl nhek the was anl nhe

16 ep:
 don't know the same thing iene ' she said to herself, 'sh toere io the marter wfre ' 'i don't know
```

Видно, что с увеличением количества эпох, улучшается качество генерации текста.

Прогресс обучения с помощью TensorBoard

Для наблюдения за процессом обучения модели был использован TensorBoard.

Графики процесса обучения представлены на рис. 1-3.

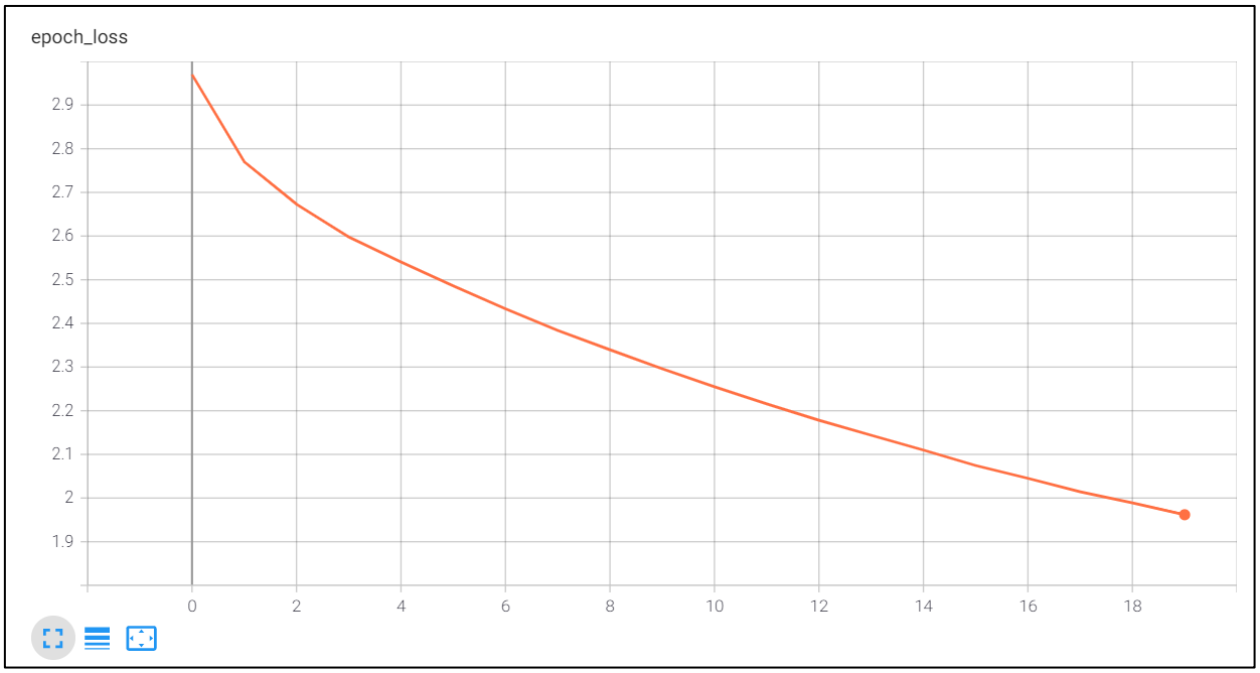

Рисунок 1 – График потерь

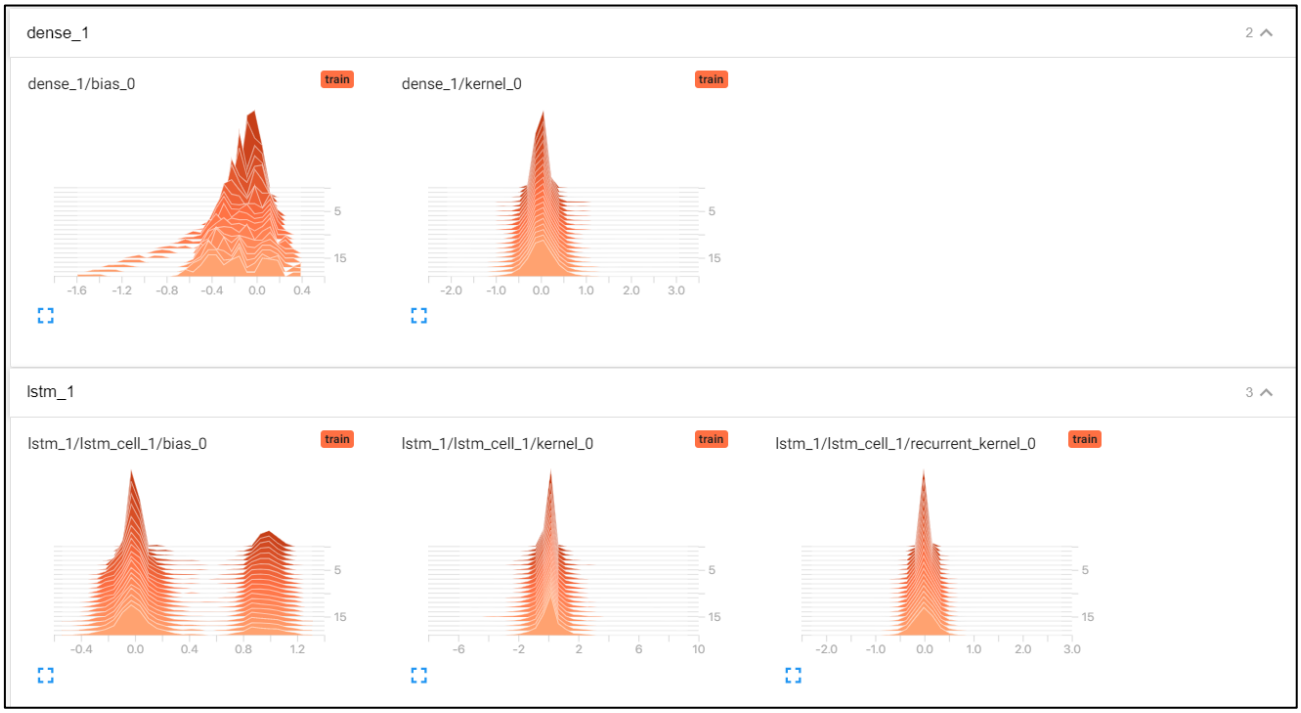

Рисунок 2 – Гистограммы активаций

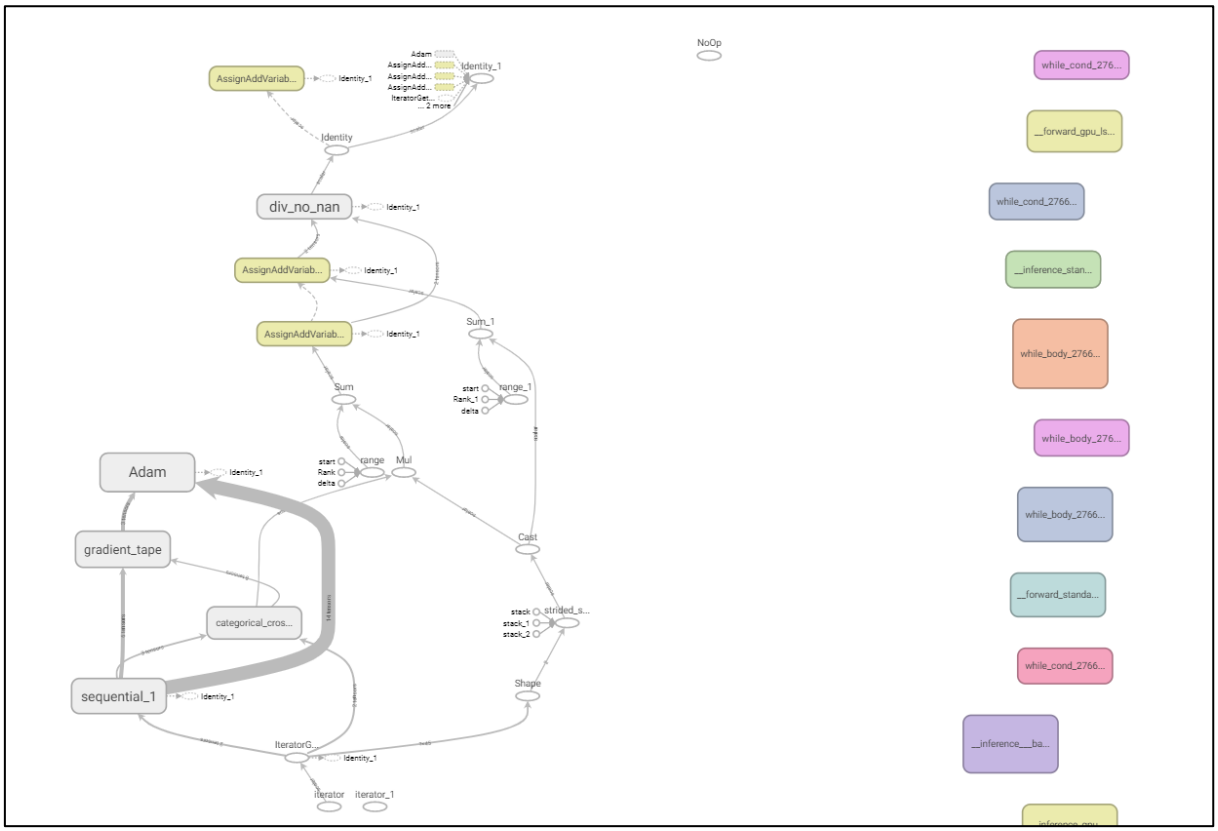

Рисунок 3 – Интерактивные диаграммы низкоуровневых операций

Выводы

В ходе лабораторной работы была реализована нейросеть для генерации текста на основе “Алисы в стране чудес”. Был также написан собственный Callback, позволяющий отслеживать прогресс обучения сети.

Приложение А. Исходный код программы

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true'
from matplotlib import gridspec
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
from keras.layers import Dense, Dropout, LSTM
from keras.models import Sequential
from keras.callbacks import ModelCheckpoint, Callback, TensorBoard
from keras.utils import np_utils

class GenerationCallback(Callback):
    def gen_text(self, size=100):
        start = np.random.randint(0, n_patterns-1)
        pattern = dataX[start]
        text = []
        for i in range(size):
            x = np.reshape(pattern, (1, len(pattern), 1))
            x = x / float(n_vocab)
            prediction = model.predict(x, verbose=0)
            index = np.argmax(prediction)
            result = int_to_char[index]
            text.append(result)
            pattern.append(index)
            pattern = pattern[1:len(pattern)]
        print("".join(text))

    def on_epoch_end(self, epoch, logs=None):
        if epoch % 5 == 0:
            print(f'{epoch+1} ep:')
            self.gen_text()


filename = "wonderland.txt"
```

```
raw_text = open(filename).read()
raw_text = raw_text.lower()

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

n_chars = len(raw_text)
n_vocab = len(chars)
print("Total Characters: ", n_chars)
print("Total Vocab: ", n_vocab)

seq_length = 100
dataX = [] # 144331 x 100
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])
n_patterns = len(dataX)
print("Total Patterns: ", n_patterns)
print(dataX[0])
print(dataY[0])

X = np.reshape(dataX, (n_patterns, seq_length, 1)) # образцы, временные шаги, признаки
X = X / float(n_vocab) # нормализация
y = np_utils.to_categorical(dataY)
# print(X.shape) # (144331, 100, 1)
# print(y.shape) # (144331, 45)
# print(X[0])
# print(y[0])
```

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

filepath="weights-improvement-{epoch:02d}.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=1, save_best_only=True, mode='min')
callbacks_list = [checkpoint, GenerationCallback(), TensorBoard(log_dir='logs', histogram_freq=1, embeddings_freq=1)]

model.fit(X, y, epochs=20, batch_size=128, callbacks=callbacks_list, verbose=2)

filename = "weights-improvement-20.hdf5"
model.load_weights(filename)

start = np.random.randint(0, n_patterns-1)
pattern = dataX[start]
print("Seed:")
print("\"", ''.join([int_to_char[value] for value in pattern]), "\"")
text = []

for i in range(100):
        x = np.reshape(pattern, (1, len(pattern), 1))
        x = x / float(n_vocab)
        prediction = model.predict(x, verbose=0)
        index = np.argmax(prediction)
        result = int_to_char[index]
        text.append(result)
        pattern.append(index)
        pattern = pattern[1:len(pattern)]
print("".join(text))
```