**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе №8**

**по дисциплине «Искусственные нейронные сети»**

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

|  |  |  |
| --- | --- | --- |
| Студент гр. 8383 |  | Сосновский Д. Н. |
| 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

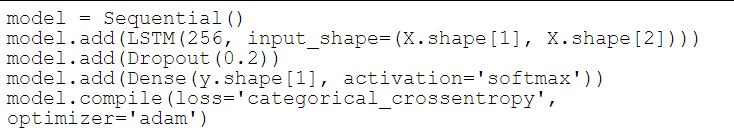
• Ознакомиться с системой Callback в Keras

**Требования**

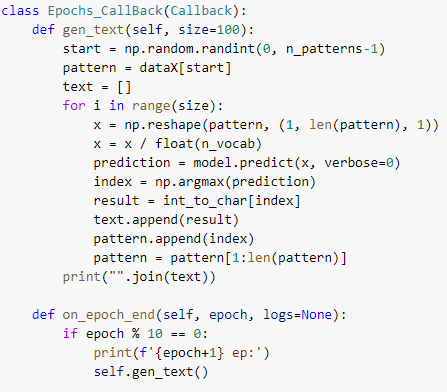
* Реализовать модель ИНС, которая будет генерировать текст
* Написать собственный CallBack, который будет показывать то, как генерируется текст во время обучения (то есть раз в какое-то количество эпох генерировать и выводить текст у необученной модели)
* Отследить процесс обучения при помощи TensorFlowCallBack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

**Ход работы.**

Была построена модель нейронной сети. Архитектура модели представлена ниже.

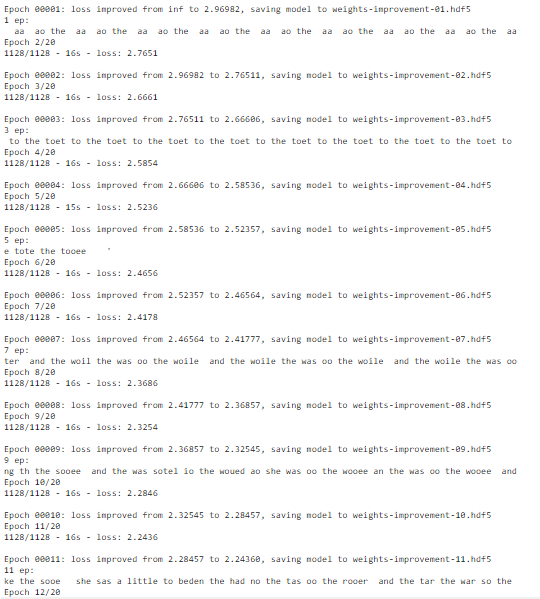


Полный исходный код программы приведён в приложении. Также был написан собственный CallBack, названный как Epochs\_CallBack. Код класса Epochs\_CallBack приведён ниже.



Данный CallBack реализует вывод сгенерированного текста на каждой второй эпохе у необученной модели.

Далее программа была запущена. Сгенерированный текст, выводимый нашим CallBack-ом, представлен ниже.



Видно, что с каждым выводом повышается качество текста.

Далее был использован TensorBoard для отслеживания процесса обучения модели.

Графики и гистограммы, полученные с помощью TensorBoard, представлены ниже.

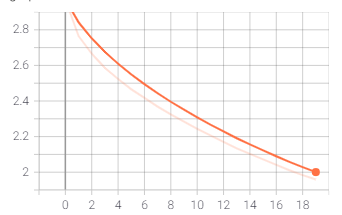


Рисунок - график потерь (epoch\_loss)

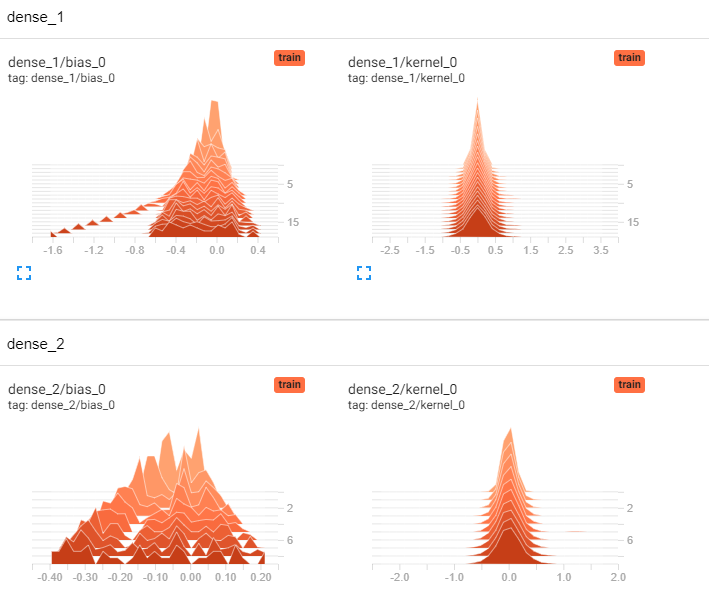


Рисунок - гистограммы активаций

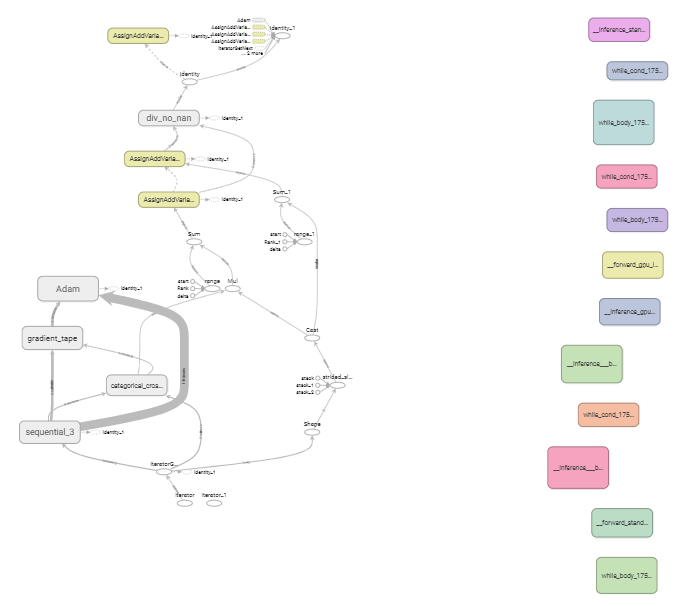


Рисунок - диаграммы низкоуровневых операций

**Вывод.**

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**

В этом приложении приведён исходный код программы.

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 Epochs\_CallBack(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 % 2 == 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)))

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)

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])

n\_patterns = len(dataX)

X = np.reshape(dataX, (n\_patterns, seq\_length, 1))

X = X / float(n\_vocab)

y = np\_utils.to\_categorical(dataY)

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, Epochs\_CallBack(), 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]

text = []

for i in range(200):

        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))