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

ЛАБОРАТОРНАЯ РАБОТА №3

по дисциплине «Искусственные нейронные сети»

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

Студент гр. 7383 _____ Лосев М.Л.

Преподаватель _____ Жукова Н.А.

Санкт-Петербург

2020

Цели.

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

Выполнение работы.

Чтобы реализовать модель ИНС, которая будет генерировать текст, выбрана арихитектура со слоем с долгой краткосрочной памятью (LSTM) и слоем Dense:

```
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'))
```

Чтобы показывать, как генерируется текст на каждой эпохе обучения, от `keras.callbacks.Callback` отнаследован класс `CustomCB`:

```
class CustomCB(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        f = open(output_filename, 'a')
        f.write("\nEpoch: "+str(epoch)+"\n")
```

```
    f.close()
    generate(model)
```

Чтобы отследить процесс обучения, используем написанный CallBack, который сохранил в файл на каждой эпохе семя генерации и сгенерированный текст. Рассмотрим результаты на некоторых эпохах.

После первой эпохи сгенерировался текст:

oe aa aa ao ao a " " " " " "' ' "'

Он не имеет смысла и представляет из себя хаотично повторяющиеся три буквы, пробелы и апострофы.

После второй эпохи:

th the kooee to the toee to the toee to the toee to the toee to the toee to the toee to the toee to the toee to the toee to the toee to the toe

Тут уже есть два слова английского языка. Текст состоит из повторяющегося шаблона. Смысла нет.

Дальше по мере обучения появляются другие слова. На 15 эпохе:

nel the qieeen of the sine of the siasee th the white rabbit in the sase thing sar the wirte tabbit she was soi thne th the white rabbit and the was soin io the could beter ate the woodd and the waide

После последнеей, 20 эпохи:

could the was a lange hare wat int ano her hind the tabdit sho gnd to the lortle gaal in the coolo, and taed to the see oft rfe was oo tar of the woide, and the wai qoiel oo the was so tee the had been toinengng to aelut thit in the dould and whnt an toeer toaee a mottle and ien si the dool of the toold a lottle oo the tay of the was oo tar of the whsl a little so bro oo the tan oo tay.

Смысла все еще нет, но текст почти на половину состоит из реально существующих слов и даже содержит некоторые грамматические конструкции языка. Можно предположить, что еще некоторое количество эпох обучения позволило бы генерировать связный текст, но одна эпоха обучения занимает 15 минут, а нужно их много, судя по всему.

Вывод.

В ходе выполнения данной работы была реализована модель ИНС, которая генерирует текст на основе книги «Алиса в стране чудес». Была изучена система CallBack в keras. После каждой эпохе был сгенерирован текст. С ростом числа эпох сгенерированный текст становится больше похож на написанный человеком, но даже после 20 эпох он далек от достаточной осмысленности.

Приложение А

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import boston_housing

def draw_plot(label, train, val, blocks_num):
    plt.clf()
    arg = range(1, len(train) + 1)
    plt.plot(arg, train, 'r', label='Training')
    plt.plot(arg, val, 'b', label='Validation')
    plt.title('Training and validation ' + label)
    plt.xlabel('Epoch')
    plt.ylabel(label)
    plt.legend()
    plt.savefig(str(blocks_num) + '_fold_' + label + '.png')

def normalize(data):
    mean = data.mean(axis=0)
    data -= mean
    std = data.std(axis=0)
    data /= std

def build_model(shape):
    model = Sequential()
    model.add(Dense(64, activation='relu', input_shape=shape))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(1))
    model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
    return model

def fold(data, targets, k, num_epochs = 100):
    num_val_samples = len(data) // k
    fold_history = {'mae': [], 'val_mae': []}
    for i in range(k):
        print('processing fold #', i)
        val_data = data[i * num_val_samples: (i + 1) * num_val_samples]
        val_targets = targets[i * num_val_samples: (i + 1) * num_val_samples]
        train_data = np.concatenate(
            [data[:i * num_val_samples], data[(i + 1) * num_val_samples:]],
axis=0)
        train_targets = np.concatenate(
            [targets[:i * num_val_samples], targets[(i + 1) *
num_val_samples:]], axis=0)
        model = build_model((train_data.shape[1],))
        H = model.fit(train_data, train_targets, epochs=num_epochs,
batch_size=1, verbose=0,
            validation_data=(val_data, val_targets))
        mae = H.history['mean_absolute_error']
        val_mae = H.history['val_mean_absolute_error']
        loss = H.history['loss']
        val_loss = H.history['val_loss']
        fold_history['mae'].append(mae)
        fold_history['val_mae'].append(val_mae)
        draw_plot('MAE (block #' + str(i) + ')', mae, val_mae, k)
    draw_plot('fold MAE',
        [np.mean([x[i] for x in fold_history['mae']]) for i in
range(num_epochs)],
        [np.mean([x[i] for x in fold_history['val_mae']]) for i in
range(num_epochs)], k)
```

```
(train_data, train_targets), (test_data, test_targets) =
boston_housing.load_data()
# All the data available is both train and test, default separation is
conditional
data = np.concatenate([train_data, test_data], axis=0)
targets = np.concatenate([train_targets, test_targets], axis=0)
normalize(data)
fold(data, targets, 5)
```