МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МО ЭВМ

ОТЧЁТ

по лабораторной работе №8

по дисциплине «Искусственные нейронные сети»

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

Студент гр.8382 _____ Фильцин И.В.

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.

Задание

Ознакомиться с генерацией текста

Ознакомиться с системой Callback в Keras

Ход работы

В качестве датасета используется текст из произведения "Приключения Алисы в Стране Чудес".

Архитектура сети:

Layer (**type**)	Output Shape	Param #
lstm (LSTM)	(None, 256)	264192
dropout (Dropout)	(None, 256)	0
dense (Dense)	(None, 45)	11565

Был написан свой callback для вывода сгенерированного моделью текста на каждой второй эпохе

```
class TaskCallback(Callback):
def on_epoch_end(self, epoch, logs=None):
```

```
if epoch % 2 == 0:
    start = numpy.random.randint(0, len(dataX) - 1)
    pattern = dataX[start]
    print("Seed:")
    print("\"", ''.join([int_to_char[value] for value in pattern]), "\"")
    for i in range(100):
        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]
        seq_in = [int_to_char[value] for value in pattern]
        sys.stdout.write(result)
        pattern.append(index)
        pattern = pattern[1:len(pattern)]
```

Результаты вызова callback:

Epoch1: toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe toe

Epoch3: nhe toe toet the woet toe toet the toet the toet the toet the toet the toet the toet the toet the

Epoch5: hare **and** the toee to the toee **and** she soee toe toee to the toee **and** she soee toe toee to the toee

Epoch7: the toeee the woel to tee toeee th the woeee the woel to tee toee 'the woie to toe to toe to

Epoch9: and the woine sab iu tas a lirtle oo the could the care woin a lirtle oo the toiee

Epoch11: and the wosld the care woue the care woue the care oo the sooee and the while tabbit was so tee the

Epoch13: the soeent oa thing 'i can t lene the soie, said the cotmouse, 'tee iitt aadin, said the cot

Epoch19: aed the seme tf the taad,' shi taid to herself, 'so the was to tie tiaeen she sas oo the tame bear

Пример сгенерированного текста уже обученной моделью:

and the was notting at the sooee of the sabbit has shter th the was of the tanee oate the tan of the tabbit tas th the wond

Вывод

В ходе лабораторной работы была реализована рекуррентная сеть для генерации текста.

Приложение А.

Исходный код

```
# Load LSTM network and generate text
import sys
import numpy
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM
from tensorflow.keras.callbacks import ModelCheckpoint
import tensorflow.keras.utils as utils

# load ascii text and covert to lowercase

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

# create mapping of unique chars to integers, and a reverse mapping

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))

# summarize the loaded data

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

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

# prepare the dataset of input to output pairs encoded as integers

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)
```

```
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 = utils.to_categorical(dataY)

# define the LSTM model
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'))

# load the network weights
filename = "weights-improvement-20-1.9484.hdf5"
model.load_weights(filename)
model.compile(loss='categorical_crossentropy', optimizer='adam')

model.summary()

# pick a random seed
start = numpy.random.randint(0, len(dataX)-1)
pattern = dataX[start]

print("Seed:")
print("\"", ''.join([int_to_char[value] for value in pattern]), "\"")

# generate characters
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]
    seq_in = [int_to_char[value] for value in pattern]
    sys.stdout.write(result)
    pattern.append(index)
    pattern = pattern[1:len(pattern)]

print("\nDone.")
```