МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МО ЭВМ

ЛАБОРАТОРНАЯ РАБОТА №8

по дисциплине «Искусственные нейронные сети»

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

Студентка гр. 7381	_________________	Машина Ю.Д.
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.

1. Реализовать модель ИНС, которая будет генерировать текст

Была выбрана модель следующего вида:

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

Код представлен в приложении.

2. 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)

Был написан MyCallback, который выводит в файл generated.txt сгенерированный текст раз в 5 эпох.

3. Отследить процесс обучения при помощи TensorFlowCallBack, в отчете привести результаты и их анализ

Содержимое generated.txt (повторы представлены в виде троеточия):

Epoch 1	e ao a a
Epoch 5	har and the woete the was ho aalin the was ho a lirt oo the toeee to the was ho aalie an the was ho aalie …
Epoch 10	bit sertee to the soite, 'ie cou t tooe to tee tou toie to tee ' said the caterpillar. 'ie you te toen the sooe,' said the caterpillar. …
Epoch 15	to the toiet of the woide ' 'io tou dre t teoe the soiee said alice. 'io whu would ' said the daterpillar. 'ie course tou doon,' said the monk turtle. 'io whu would ' said the daterpillar. …
Epoch 20	be no gren the tiater was an thll as she cadl to the while sabbit war so the tooe as the was anong th the taate th the tase tire bio the hareen oo the tase wirh sheee an shle the had so tooe tas anong the thaee of the was and the was ano the tabbit wire aidin, and the whit hlr she was allie to tiing that she was at thl fad ne the was anl the war ani the was ani the was ani …
Epoch 25	e hirtt wott on toe tose, and the white rabbit was soi oittle gorge on ter that she was soa theee an thll the houpe so toiekl at the cadl if the hirst woile sa tooe the had sever oo tee sfat the had been whit sire the had been whit sas ao hnr oote, and sas

	no alice a garge carl and thit was so tiee th the thate sas ao once tf thie. 'the huehtsd then it was ' shi gaden in a lott of great auuiari toine and hor aedin to the thaee of the dourd sh the garter to the thre the had buew woth the wes of thr whrh the was she whst sote tf toee afd '-- …
Epoch 30	herterf to the whyt sitenk, and the theted sae it ala not thieg oefe the had so ere the had bnene thth the courd bnd the had selenken the had bnent thet war to tinl wer oo the could so the whol the had sedene the had so toe oh the goore of the coor wo the cool to the coor wotteng the was solntiigg tore thted of the coorars of the shreok tilt har iand. 'toele wou dll tou,' said the kork taid, and the whit har to the soreo of the tereon afout the tas oo thrt soreo. what sorld be to tee it whi mucer willese!' taid the koygh tirr pame, ''thll, i shanl ho whe wante tales in the teathas.' 'tol don't sioed toere'' said the kork. ''i ve teen thet woued ' the said to herself, 'at thas aad toert in the tore. 'ie you don't know it,' said alice angil.y ard tfreoning tone. 'a dours wfin i senely hotw the toodl of that io woen the samter of that moke to then tou doene toene of the way soe ofde of the sabeit. and the hotke of the thre the qabbit was so toeke tas soene on the sooe of the care s

Существуют две основные дихотомии при использовании RNN для для текста: символьная и словесная. Мы изучаем последовательности символов. Как видно, с каждой новой эпохой текст становится более связным, однако многие символы не объединились в существующие слова. 30 эпох, очевидно, недостаточно для того, чтобы получить более связный текст, но больше эпох заняло бы очень много времени.

Вывод.

В ходе выполнения данной работы было произведено ознакомление с генерацией текста и системой Callback в Keras.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

```
# LSTM and CNN for sequence classification in the IMDB dataset
import string

import numpy as np
import datagen as datagen
#from keras import
from keras.callbacks import ReduceLROnPlateau
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM, GRU, Dropout
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
import matplotlib.pyplot as plt
from keras.layers import *
from sklearn.ensemble import AdaBoostClassifier
# Import Support Vector Classifier
from sklearn.svm import SVC
# Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
import numpy as np
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier

def singleModelPlots( histories ):
    #title = []
    for history in histories:
        plt.plot(history.history['accuracy'])
        plt.plot(history.history['val_accuracy'])
        plt.title('Accuracy')
        plt.ylabel('Accuracy')
        plt.xlabel('Epoch')
        plt.legend(['Train', 'Test'], loc='upper left')
        plt.show()

        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
        plt.title('Loss')
        plt.ylabel('Loss')
        plt.xlabel('Epoch')
        plt.legend(['Train', 'Test'], loc='upper left')
        plt.show()
    return

def justPlots( history ):
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()

    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()
    return

def build_CNN_model():
    model = Sequential()
    model.add(Embedding(top_words, embedding_vector_length,
input_length=max_review_length))
    model.add(Conv1D(filters=32, kernel_size=3, padding='same',
activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    #model.add(LSTM(100, recurrent_dropout=0.2))
    model.add(Dense(1, activation='sigmoid'))
    #model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
    return model

def build_model():
    model = Sequential()
    model.add(Embedding(input_dim=top_words,
output_dim=embedding_vector_length, input_length=max_review_length))
    model.add(Conv1D(filters=32, kernel_size=3, padding='same',
activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Conv1D(filters=32, kernel_size=3, padding='same',
activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Dense(256, activation='relu'))
    model.add(Conv1D(filters=32, kernel_size=3, padding='same',
activation='relu'))
    model.add(LSTM(128, recurrent_dropout=0.3))
    #model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))
    return model

def load_data(dimension=10000):
    (training_data, training_targets), (testing_data,
testing_targets) = imdb.load_data(num_words=top_words)
    # truncate and pad input sequences

    training_data = sequence.pad_sequences(training_data,
maxlen=max_review_length)
    testing_data = sequence.pad_sequences(testing_data,
maxlen=max_review_length)

    data = np.concatenate((training_data, testing_data), axis=0)
    targets = np.concatenate((training_targets, testing_targets),
axis=0)

    test_x = data[:10000]
    test_y = targets[:10000]
    train_x = data[10000:]
    train_y = targets[10000:]


    test_x = data[:10000]
    print(test_x.shape)
    test_y = targets[:10000]
    train_x = data[10000:]
    print(train_x.shape)
    train_y = targets[10000:]
    return (test_x, test_y, train_x, train_y)
```

```
def test_my_text(filename, dimension=10000):

    text = []
    with open(filename, 'r') as f:
        for line in f.readlines():
            text+=line.translate(str.maketrans('', '',
string.punctuation)).lower().split()
    indexes = imdb.get_word_index() # use ready indexes
    print(indexes)
    print(text)
    encoded = []
    for word in text:
        if word in indexes and indexes[word] < 10000: # <10000 to
avoid out of bounds error
            print('found '+word+' in indexes. its index is '+
str(indexes[word]))
            encoded.append(indexes[word])
    print('---------------------')
    print(np.array(encoded))

    reverse_index = dict([(value, key) for (key, value) in
indexes.items()])

    decoded = " ".join([reverse_index.get(i , "#") for i in
np.array(encoded)]) # не пон почему в ориге i-3
    print(decoded)
    test_x, test_y, train_x, train_y = load_data()

    print(decoded)
    #print(len(text.split()))
    model = build_model()
    model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
    model.fit(train_x, train_y, epochs=2, batch_size=200,
validation_data=(test_x, test_y))
     # vectorize just like we did with data
    #print(model.predict(vectorize([np.array(encoded)])))

    return model.predict(sequence.pad_sequences(np.array(encoded),
maxlen=max_review_length))

# fix random seed for reproducibility
np.random.seed(7)
# load the dataset but only keep the top n words, zero the rest
```

```
top_words = 10000
max_review_length = 500
if __name__ == '__main__':
    (training_data, training_targets), (testing_data,
testing_targets) = imdb.load_data(num_words=top_words)
    # truncate and pad input sequences

    training_data = sequence.pad_sequences(training_data,
maxlen=max_review_length)
    testing_data = sequence.pad_sequences(testing_data,
maxlen=max_review_length)

    data = np.concatenate((training_data, testing_data), axis=0)
    targets = np.concatenate((training_targets, testing_targets),
axis=0)

    test_x = data[:10000]
    test_y = targets[:10000]
    train_x = data[10000:]
    train_y = targets[10000:]
    # create the model
    embedding_vector_length = 32

    #model = build_RNN_model()

    batch_size = 64
    epochs = 2


    """
     kfold = model_selection.KFold(n_splits=10, random_state=13)
        # create the sub models
        estimators = []
        model1 = LogisticRegression()
        estimators.append(('logistic', model1))
        model2 = DecisionTreeClassifier()
        estimators.append(('cart', model2))
        model3 = SVC()
        estimators.append(('logistic2', model3))
        # create the ensemble model
        ensemble = VotingClassifier(estimators)
        ensemble.fit(train_x, train_y)
        print(ensemble.score(test_x, test_y))
    """
    #results = model_selection.cross_val_score(ensemble, train_x,
train_y, cv=kfold)
   # print(results.mean())
    from sklearn.base import TransformerMixin
    from sklearn.datasets import make_regression
    from sklearn.pipeline import Pipeline, FeatureUnion
    from sklearn.model_selection import train_test_split
    from sklearn.ensemble import RandomForestRegressor
    from sklearn.neighbors import KNeighborsRegressor
    from sklearn.preprocessing import StandardScaler,
PolynomialFeatures
    from sklearn.linear_model import LinearRegression, Ridge
    from keras.models import Model
    from keras.layers import concatenate


   # model1 = build_CNN_model()
   # model = build_model()
    '''
    print("model1:")
  #  model1.summary()
    print("model2:")
   # model2.summary()
    merged_layers = concatenate([model1.output, model2.output])
    x = BatchNormalization()(merged_layers)
    x = Dense(300)(x)
    x = PReLU()(x)
    x = Dropout(0.2)(x)
    x = Dense(1)(x)
    x = BatchNormalization()(x)
    out = Activation('sigmoid')(x)
    merged_model = Model([model1.input, model2.input])
    print("merged model:")
    #merged_model.build(10000)
    #merged_model.summary()
    merged_model.compile(loss='binary_crossentropy',
optimizer='adam', metrics=['accuracy'])
    #print(train_x[0])
    '''

   # model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

```
   # history = model.fit(train_x, train_y, validation_data=(test_x,
test_y), epochs=epochs, batch_size=batch_size)
    #justPlots(history)

   # X, y = make_regression(n_features=10, n_targets=2)
   # X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

   # №model.fit(train_x, train_y)
    #score = model.score(test_x, test_y)

    print('Done. ')

    from sklearn.linear_model import LogisticRegression
    print('logistic regression:')
    # create a new logistic regression model
    log_reg = LogisticRegression()
    # fit the model to the training data
    log_reg.fit(train_x, train_y)
    print((log_reg.score(test_x, test_y)))

    print("test1:")
    print(str(test_my_text('test1.txt')))
    print("test2:")
    print(str(test_my_text('test2.txt')))
    print("test3:")
    print(str(test_my_text('test3.txt')))
    print("test4:")
    print(str(test_my_text('test4.txt')))
    print("test5:")
    print(str(test_my_text('test5.txt')))

   # history = model.fit(train_x, train_y, validation_data=(test_x,
test_y), epochs=epochs, batch_size=batch_size)
'''
    model = []
    model.append(build_RNN_model())
    model.append(build_CNN_model())

    models = []
    history = []
    for i in range(len(model)):
        i_history = model[i].fit(train_x, train_y,
validation_data=(test_x, test_y), epochs=epochs,
batch_size=batch_size)
        models.append(model[i])
        history.append(i_history)
        print(model[i].summary())
        scores = model[i].evaluate(test_x, test_y, verbose=0)

        print("Accuracy: %.2f%%" % (scores[1] * 100))
        print("Accuracy: %.2f%%" % (scores[1] * 100))
        #singleModelPlots(i_history)
'''


"""
   mergedOut = Add()([model1.output, model2.output])
    #mergedOut = Flatten()(mergedOut)
    mergedOut = Dense(256, activation='relu')(mergedOut)
    mergedOut = Dropout(.5)(mergedOut)
    mergedOut = Dense(128, activation='relu')(mergedOut)
    mergedOut = Dropout(.35)(mergedOut)

    # output layer
    mergedOut = Dense(1, activation='softmax')(mergedOut)

    from keras.models import Model

    newModel = Model([model1.input, model2.input], mergedOut)
    newModel.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

    from sklearn.ensemble import AdaBoostClassifier

    model1 = AdaBoostClassifier(random_state=1)
    train_history = model1.fit(train_x, train_y)
    test_history = model1.score(train_x, train_y)
    #merged_history = newModel.fit([train_x, train_y],
validation_data=(train_x, train_y), epochs=epochs,
batch_size=batch_size)
    #newModel.summary()
    #print(newModel.evaluate(test_x, test_y, verbose=0))
    #justPlots(merged_history)
```

```
 #print(history.history)
 justPlots(train_history)
 justPlots(test_history)
"""
```