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»

Студент гр. 7381 _____ Минуллин М.А.

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

Ход работы.

Была реализована модель ИНС, которая будет генерировать текст.

Код представлен в приложении. Модель:

```
model = Sequential()
model.add(LSTM(1, 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')
```

Был написан 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 epochCallback, который выводит в файл generated_text.txt сгенерированный текст каждую эпоху.

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

Содержимое файла generated_text.txt:

Эпоха	Сгенерированный текст
1	tee toe toe toe toe toe toe toe toe toe toe toe toe…
2	the and the and the and the and the and the…
3	ee to tee toee to the toee to the toee to the toee…
4	re the woeee to the woeee to the woeee and the woeee…
5	the toee " '
6	he woeee to the toeee and the soeee to the toeee and the woele tas io the woeee…
7	to tee to tee to tee to tee to tee so the wouee and the was anl the wooee to tee toeee so the wast oo the waster…
8	and the was toen i sonee the woiee the was anl the san an the wooee the was an in tese the was an in sese the was an in sese the was an in sese "
9	o tee soeer an the care and the whs ger aelin th the soeee she was soe wo tee soree af in tie was an in was an in was aadut the wosed the was soe to the soeer oa the sas she was an in was an in was aadut the wosed the was soe to the soeer of the sasee haree aad not the whst hnr ano ano the sas an the wooee 'whe wosee to tee toee a aet to tee toeee the carter ser all toee a lirtle so tee thet she was an in wise the sas an the wooee 'whe eors sh the sore tf the soeer of the saster ' 'whu i mene the soee sarter whr oo toe
10	to the soeee the woued to tee thet so tee io the sooee the soree of the soree tf the thnt har in the sooee the hors oa the tase

	the horshon oa toene to the thet the was oo the toeee the hors oa thet io the sooer oo the soiee ' 'the woute to tee it ho ' said the kong, 't an a lote to toe to tee the horse ai ceg toe woiee to toee to the thet ' 't al a latter to toee to tee in ' said the kong, 't a lott tait oo toe bane th toe to tee the sooer ' said the kong, 't a lott tait to tou ' said the kong, an i vool toe to tee io the sooer ' 'that so toe to toee to tee it ' said the kong, an i vool toe to tee io the sooel ' …

Так как мы учим модель последовательностям символов, многие сгенерированные слова на первых эпохах получаются бессмысленными. Можно добавить return_sequences=True в LTSM, чтобы улучшить результаты, а также увеличить кол-во эпох. Но такая сеть на моей машине будет обучаться очень долго.

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)
"""
```