Homework 3

Yixing Chen

1. General Information

1.1 Purpose

The purpose of this assignment for me is to explore text classification problem. The detailed target is to find if a review of a movie is positive or negative, e.g. If the review is “This movie is fantastic! I really like it because it is so good!”, this review should be judged as a positive review. This report explains how this type of text classification is realized.

1.2 Dataset

The dataset I use for this assignment is 50,000 IMDB movie reviews. This dataset is always used for sentiment analysis beginners to start deep learning in the past. Sentiment analysis is also a part of text classification. Hereby I use this dataset to learn this method, too. IMDB is a movie information website. Users could present their thoughts and reviews for the movie they watched. The dataset is downloaded from http://ai.stanford.edu/~amaas/data/sentiment/. All the reviews of this dataset are binary sentiment classification. Pre-processing is implemented to the dataset. The text reviews are labelled as 1 or 0 for positive or negative judgment, respectively. After pre-processed, the 50,000 reviews are saved in the file *movie_data.csv*. Each review has a corresponding binary label.

```
import pandas as pd
import numpy as np
import tensorflow as tf
data = pd.DataFrame()
data = pd.read_csv('movie_data.csv', encoding='utf-8')

X_train = data.loc[:24999, 'review'].values
y_train = data.loc[:24999, 'sentiment'].values
X_test = data.loc[25000:, 'review'].values
y_test = data.loc[25000:, 'sentiment'].values
```

This code imports dataset from *movie_data.csv*. And the dataset is split into two subsets. Among them, 25000 reviews are used as training dataset and the other 25000 reviews are used as testing dataset.

2. Main Architecture

In this assignment, text classification problem is explored in the following steps:

2.1 Word Embedding

Word Embedding used in this assignment is generated. All reviews need to be encoded as a unique representation. The text corpus of the reviews is vectorized after turning each text into a sequence of integers. The Python package used here is Keras. The following code is used to realize this function. The code provides each English word a corresponding integer, e.g. 1 represents 'the'. 3 represents 'and'. 806 represents 'check', etc. Keras includes 125,601 English words and their integers. Thus, all IMDB reviews are transformed into tokens after punctuation and space are removed. After all the words are transformed into integers, every review is converted to a vector of integers. All the vectors are defined to have the same length. The length is *(max_length)*, which is the length of the text which has most words. In this assignment, this value equals 2678. For the vector whose length is less than 2678, '0's are supplemented before it.

```
from tensorflow.python.keras.preprocessing.text import Tokenizer
from tensorflow.python.keras.preprocessing.sequence import pad_sequences

tokenizer_object = Tokenizer()
total_reviews = X_train + X_test
tokenizer_object.fit_on_texts(total_reviews)

# pad sequences
max_length = max([len(s.split()) for s in total_reviews])
#max_length=500

# define vocabulary size
vocab_size = len(tokenizer_object.word_index) + 1

X_train_tokens =    tokenizer_object.texts_to_sequences(X_train)
X_test_tokens = tokenizer_object.texts_to_sequences(X_test)


X_train_pad = pad_sequences(X_train_tokens, maxlen=max_length, padding='pre')
X_test_pad = pad_sequences(X_test_tokens, maxlen=max_length, padding='pre')
```

The vocabulary size *(vocab_size)* of network input is the number of all vocabulary in Keras plus 1. The dimension of embedding layer *(embedding_dim)* is set to 100.

2.2 Building Model

The neural network model used in this assignment is sequential. This means this model passes a list of layer instances to the constructor. The first layer of this model is to receive information about its input shape. Hereby the first hidden layer of this neural network is embedding layer. The input length is *(max_length).* The embedding dimension is set to 10. The shape of output of the model is (None, 10, 64), where None is the embedding dimension. The second layer is LSTM layer, which is different from the model of homewor1. To make this report concise, the main principle of LSTM is not given here. The second hidden layer contain 32 units of nodes. The third layer is a dense layer, which contains one unit here. The initial weights of the network are randomly set. The weights will be updated with the network training process. The number of parameters of the model trained is 12,573,001. Among these parameters, 12,560,200 are for embedding layer, which is the result of *vocab_size* multiplied by *embedding_dim*. 12,768 are for LSTM network's second later. 33 are for the merged one dense node. The activation function used in this network is sigmoid function.

```
Summary of the built model...
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_7 (Embedding)      (None, 2678, 10)          1256020
_________________________________________________________________
lstm_7 (LSTM)                (None, 32)                5504
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,261,557
Trainable params: 1,261,557
Non-trainable params: 0
_________________________________________________________________
```

```
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, GRU
from keras.layers.embeddings import Embedding

EMBEDDING_DIM = 100

print('Build model...')

model = Sequential()
model.add(Embedding(vocab_size, EMBEDDING_DIM, input_length=max_length))
model.add(LSTM(units=32,    dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))
```

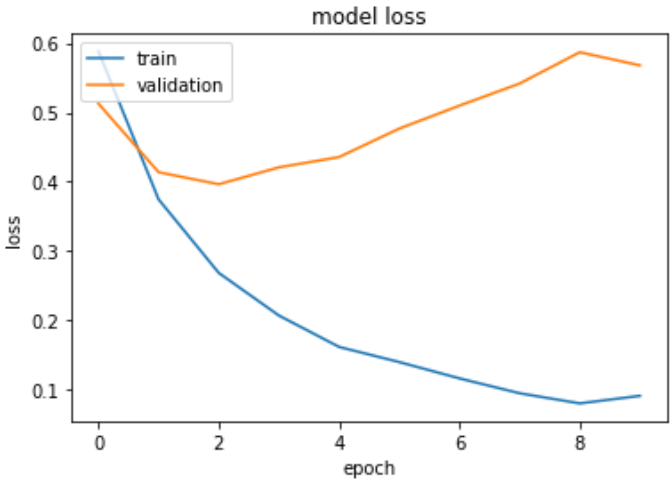
2.3 Training Model

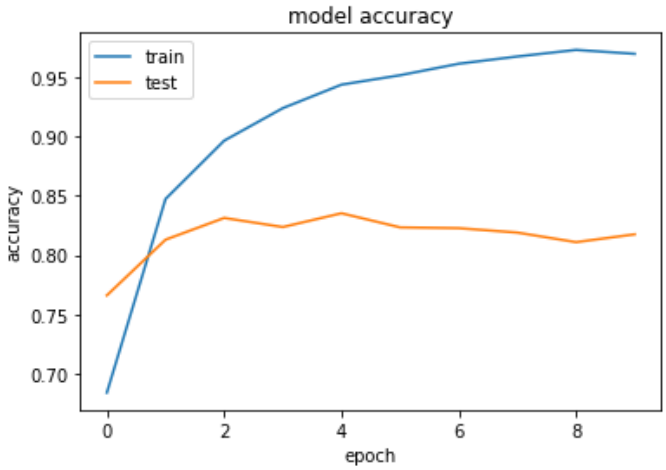
The neural network model is trained with training dataset and the 25,000 text dataset mentioned beforehand is used to validate the accuracy of the model. The training method of this model is *Adam*, which is a stochastic optimization algorithm. The details of this algorithm could be found in the paper https://arxiv.org/abs/1412.6980v8. The batch size is set to 128. The training epoch is set to 10. The learning rate is the default value, 0.001. The loss function is binary cross entropy, which means the error between real value and estimated value is 0 or 1.

```
# try using different optimizers and different optimizer configs
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

print('Summary of the built model...')
print(model.summary())

print('Train...')

history = model.fit(X_train_pad, y_train, batch_size=128, epochs=10, validation_data=(X_test_pad, y_test), verbose=2)

print(history.history.keys())
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()

plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

The training process is shown as the following:

```
Train...
Train on 25000 samples, validate on 25000 samples
Epoch 1/10
 - 606s - loss: 0.5881 - acc: 0.6838 - val_loss: 0.5127 - val_acc: 0.7660
Epoch 2/10
 - 595s - loss: 0.3744 - acc: 0.8474 - val_loss: 0.4136 - val_acc: 0.8130
Epoch 3/10
 - 597s - loss: 0.2683 - acc: 0.8966 - val_loss: 0.3963 - val_acc: 0.8314
Epoch 4/10
 - 599s - loss: 0.2067 - acc: 0.9241 - val_loss: 0.4207 - val_acc: 0.8237
Epoch 5/10
 - 602s - loss: 0.1614 - acc: 0.9440 - val_loss: 0.4355 - val_acc: 0.8353
Epoch 6/10
 - 604s - loss: 0.1396 - acc: 0.9519 - val_loss: 0.4766 - val_acc: 0.8234
Epoch 7/10
 - 597s - loss: 0.1161 - acc: 0.9616 - val_loss: 0.5098 - val_acc: 0.8227
Epoch 8/10
 - 564s - loss: 0.0946 - acc: 0.9678 - val_loss: 0.5416 - val_acc: 0.8190
Epoch 9/10
 - 553s - loss: 0.0800 - acc: 0.9733 - val_loss: 0.5869 - val_acc: 0.8109
Epoch 10/10
 - 553s - loss: 0.0908 - acc: 0.9700 - val_loss: 0.5679 - val_acc: 0.8175
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```





2.4 Validating Model

To validate the model, 25,000 test reviews are utilized. The batch size is 128. The

metrics to be evaluated by the model during training and testing is accuracy. The accuracy is 81.75% after 10 epochs. Compared to the accuracy of GRU network in homework1, which was 79.74%, the accuracy of LSTM is slightly higher.

```
print('Testing...')
score, accuracy = model.evaluate(X_test_pad, y_test, batch_size=128)

print('Test score:', score)
print('Test accuracy:', accuracy)

print("Accuracy: {0:.2%}".format(accuracy))
```

The result is shown as the following:

```
Testing...
25000/25000 [==============================] - 64s 3ms/step
Test score: 0.5679277250671386
Test accuracy: 0.8175199999809265
Accuracy: 81.75%
```

The experiment protocol here is to give some test samples to the model and check the result after classification. The test samples are some text reviews.

```
test_sample_1 = "This movie is fantastic! I really like it because it is so good!"
test_sample_2 = "Good movie!"
test_sample_3 = "Maybe I like this movie."
test_sample_4 = "Not to my taste, will skip and watch another movie"
test_sample_5 = "if you like action, then this movie might be good for you."
test_sample_6 = "Bad movie!"
test_sample_7 = "Not a good movie!"
test_sample_8 = "This movie really sucks! Can I get my money back please?"
test_samples = [test_sample_1, test_sample_2, test_sample_3, test_sample_4, test_sample_5, test_sample_6, test_sample_7, test_sample_8]

test_samples_tokens = tokenizer_obj.texts_to_sequences(test_samples)
test_samples_tokens_pad = pad_sequences(test_samples_tokens, maxlen=max_length)

#predict
model.predict(x=test_samples_tokens_pad)
```

The classification result is shown as the following:

```
array([[0.4996954 ],
       [0.011913  ],
       [0.01285431],
       [0.00146982],
       [0.02561212],
       [0.00126237],
       [0.0087561 ],
       [0.00061163]], dtype=float32)
```

Original real test dataset is used to compare the model prediction values to test the accuracy.

```
classes = model.predict(X_test_pad[:10], batch_size=128)
for i in range (0,10):
    if(classes[i] > 0.5 and y_test[i] == 1 or (classes[i] <= 0.5 and y_test[i] == 0)):
        print( classes[i], y_test[i], " Right prediction")
    else :
        print( classes[i], y_test[i], " Wrong prediction")
```

The comparison result is shown as the following. 10 pairs are selected. Among them, 1 prediction is wrong and other 9 predictions are right.

```
[0.9225818] 1  Right prdiction
[0.00824559] 1  Wrong prdiction
[0.9944743] 1  Right prdiction
[0.01884729] 1  Wrong prdiction
[0.99500364] 1  Right prdiction
[0.9957051] 1  Right prdiction
[0.91922647] 1  Right prdiction
[0.00371358] 1  Wrong prdiction
[0.9312982] 1  Right prdiction
[0.99429554] 1  Right prdiction
```