1 IMDB

1.1 Preprocessing

In the *train_NLP.py* file, firstly load the original dataset from *./data/aclImdb* folder and create two empty lists to store positive samples and negative samples as they come from different files. Then use the *gen_set* function to generate each sample from */train/pos* and */train/neg* respectively and put them into the corresponding positive list or negative list (using *.append()*) method. Combine the positive list and negative list to form the original training set. After having the original training set, the training labels are created by *np.ones* and *np.zeros*, in which positive samples obtained labels 1 and negative samples obtained labels 0. 20000 was chosen as the vocabulary size parameter, meaning that only the first 20000 words with the highest frequency would be retained. And 100 is the max length size parameter, that is to say, every sample would have the same length, 100 words. Then, conducting text vectorization, using *texts_to_sequences* method from *Tokenizer* to transfer the text into numbers. And as some sentences are shorter than 100 words themselves or some sentences contain less than 100 words after filter the low-frequency words, the padding step is conducted to the obtained sequences. In the *test_NLP.py* file, the same preprocessing steps are implemented to the testing set. In the text vectorization procedure, as testing samples need to be fit in the same tokenizer as the training samples, the tokenizer is generated again in *test_NLP.py* file.

1.2 Model

The first layer is an *Embedding* layer using 32 as the word vector dimension parameter. *Embedding* layer provides the same effect as a fully connected layer who has conducted

one-hot encoding to the input data, thus improving training efficiency. As the output of the *Embedding* layer is a 2D vector, in which each word in the input sequence of words has one embedding. A *Flatten* layer is added after the *Embedding* layer to flatten the 2D matrix into a 1D vector to feed into the next *Dense* layer. After the *Embedding* and *Flatten* layers, a *Dense* layer with 256 units using *Relu* activation function is added. And finally, the output layer with 1 units using *Sigmoid* function is added to the NLP model to complete the binary classification task.

Totally 4 epochs are run on training set, with a 512 batch size using *adam* optimizor and *binary_crossentropy* loss function. And the model is saved as 20862738_NLP_model.h5.

1.3 Accuracy

The final training accuracy reaches 98.7% after 4 epochs which seems kind of overfitting, but the testing accuracy is around 84% which is not very bad.