Jun Sung Kim
CMPSC 448 Final Project

The task for this final project is to use both CNN and RNN for IMDB movie sentimental classification. The dataset is loaded from Kaggle IMDB movie review competition. The dataset contains two variables, Review, and binary sentiment classification (positive or negative). Review contains movie reviews in text. A possible task would be to predict the number of positive and negative reviews using both CNN and RNN algorithms. Since the dataset contains text data, it is necessary to convert it into meaningful numerical values. We can apply one-hot-encoding for the sentiment variable accordingly. To test how the model performs, I decided to split them into train and test data before I clean the data. I splitted the data into half 0.5).

The next step for preprocessing would be to convert text data into numerical values. I started with tokenization. The built-in Tokenization function from TensorFlow package will allow us to achieve this task. The tokenization process was necessary because RNN and CNN requires the text data to be tokenized so that each word can be used in meaningful way. Tokenization is required for both the train and test dataset. The next step is to convert tokenized dataset to integers.

Tokenizer.texts_to_sequences function will convert each word into mapped integers. If we do this for both the train and test dataset, then we will be able to successfully convert them into integers.

The final step of preprocessing is to set the maximum length of dataset that has different length. When I checked the maximum length of train data, the length of maximum sequence length was 2493. However, using the maximum length for 2493 may be too much. Setting maximum length of 500 would not harm too many samples. 92% of samples have a length less than 500 according to the code that sets the threshold. Thus, using maximum length of 500 would be sufficient to pad the dataset.

The preprocessing step is going to be the same for both models. However, the main difference would be how we decide to define the model. This is because CNN extract characteristics (Convolution) of data and analyze the patterns. Thus, the main process of CNN algorithm is following Convolution process and Pooling process. By adding the convolution process into our model, it will create only one field which is Convolution layer with fixed depth, width, and height. Finally, the pooling process in the model will allow us to select the most representative features.

Meanwhile, the RNN algorithm works in different ways. It is an essentially useful tool to analyze sequential data. It will sequentially receive each word in order by applying a recurrence formula. It weighs each stored data based on how important it is and then moves on to the next data. However, there is an issue about using RNN algorithms of "long-term dependencies". Thus, in RNN model, it was better to add Long Short-Term Memory. It uses a gate mechanism to decide which information to keep and leave. The "Relu" activation function

Fitting the CNN model also starts with sequential model. The functions that we need to use to fit the model are embedding, Convolution, Pooling, Dropout, Flatten, and Dense. First, embedding contained three parameters, total words, embedding dimensions, and the input length. Total words will allow to use only words that has maximum of 5000 words. Embedding dimension is the size to convert to the hidden vectors. The input length would be the length of the sequence which is defined in the padding process. Since the embedding vector is 32, we build each layer into 500 x 16 dimensions. Each filter's length is set to be 7 with a value of 32. The output shape would be 494 x 32. Since the filter in text analysis only moves along up and down, we use Conv1D class. Pooling step involves using Max-Pooling technique. I've decided to set 5 as a pooling size.
Regularization is necessary to prevent overfitting. "Dropout" layer is the fraction of input units to drop when we train. Rate of 0.5 means we will drop half in random to zero. The Flatten layer would the multi-dimensioned input into one dimensional array before we make the final prediction. Finally, there are two dense layers added into the model. The first dense layer is learnt about the complexity of flattened output, meanwhile dense with "sigmoid" provide binary classification output.
When I compile the model, we must use binary cross entropy because this is a binary classification task.

On the other hand, RNN model follows same procedure until how we preprocess the data. However, there are slightly different classes we must define within the modeling part. Instead of using Convolution and pooling method, simply adding LSTM layer will fit the RNN model. The reason I am using LSTM layer instead of simpleRNN is because simpleRNN works better for a time series data. For the sake of this project, LSTM will work better for dealing with the sequence data. Thus, every other function will be the same as CNN model, but I used LSTM instead of Conv1D and maxpooling.

The results could be evaluated using both accuracy and loss. The accuracy of both CNN and RNN model provided was approximately 86 to 87 percent with loss of about 0.42 in the test dataset. The accuracy went up as the number of epochs went up and the loss decreased as epochs increased. These two algorithms performed well to this IMDB sentiment classification problem and thus this model could work on any other text classification problem in the future.

Even though this model works well, there could be improvements that we can make. One of the challenges that I had faced was deciding how to control the embedding dimensions from the given dataset because it has an influence on the model, but the user must decide what number to use. I had to consider each word corresponds to a different word in the vocabulary. Thus, using high dimensional might be inappropriate. Also, given the model complexity, I thought my model is relatively simple so using 16 might be enough.