CNN Model

CNN Model is in: HW2.ipynb

In the guideline, it reads the entire file, loads it into a pandas DataFrame and outputs the required rows to a CSV file. This took too long to read and load, and my computer kept frozen during this process, so I modified it to the current version. The current version reads only the required number of rows, and extracts the top few samples of each sentiment.

For the rest of the program, I did not change anything, only changed some syntax errors.

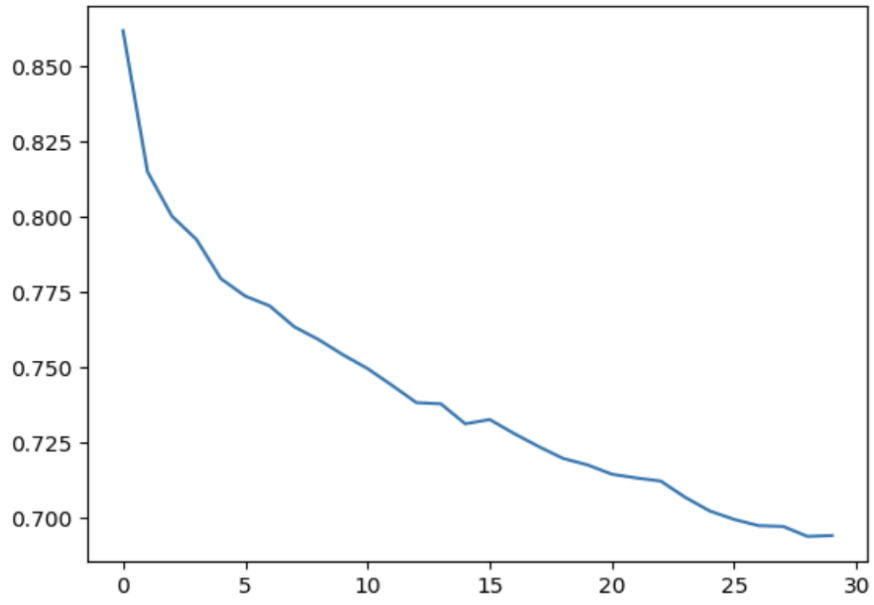
For the training process, it took about 30 minutes per epoch which is about more than 10 hours for total 30 epochs, so I move my programs to google colab, and the final result that I get is present as follow:

```
Index(['iter', ' loss'], dtype='object')
              precision    recall  f1-score   support

           0       0.76      0.77      0.76      2992
           1       0.66      0.54      0.60      3044
           2       0.72      0.84      0.77      2964

    accuracy                           0.72      9000
   macro avg       0.71      0.72      0.71      9000
weighted avg       0.71      0.72      0.71      9000

Index(['iter', ' loss'], dtype='object')
```



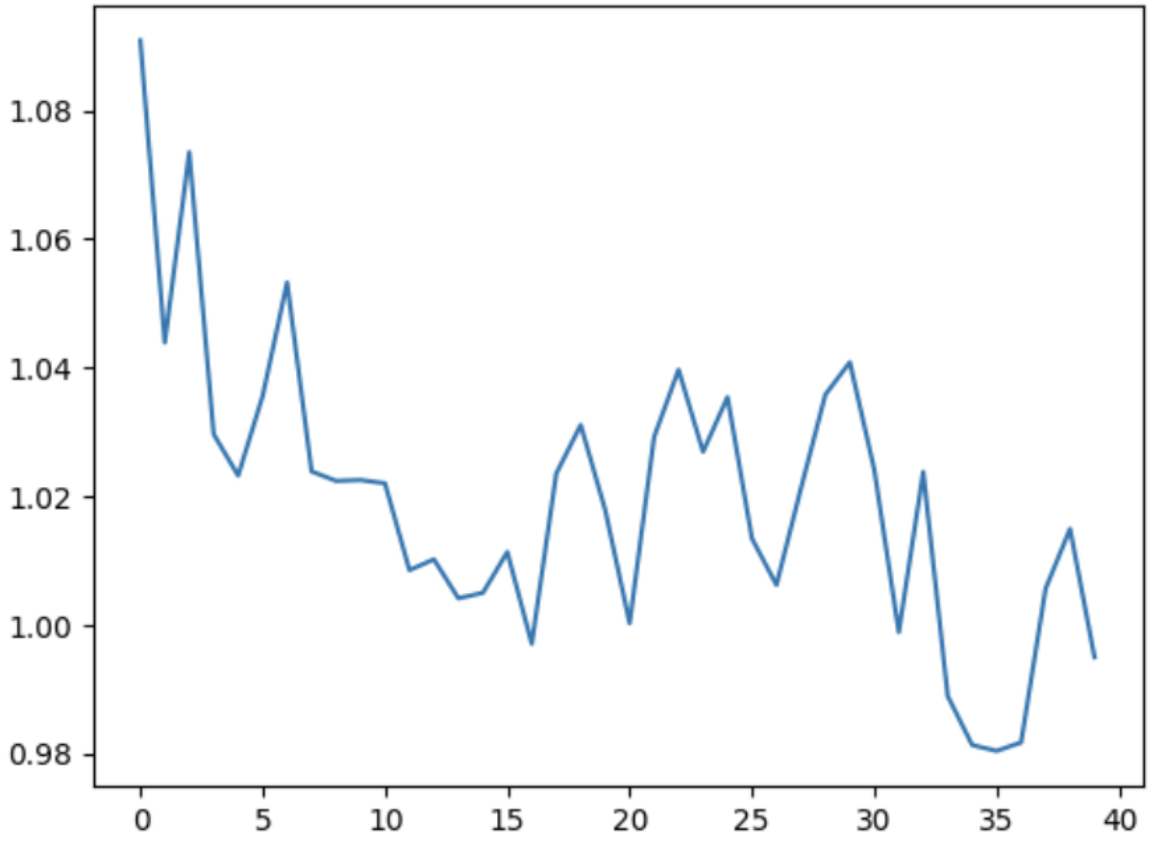
I also try it with different hyperparameters, for the result that I provide below, I change the epoch to 40 and the learning rate to 0.01 which affects the accuracy a lot. So the original model has better performance.

```
Index(['iter', ' loss'], dtype='object')
              precision    recall  f1-score   support

           0       0.61      0.72      0.66      2992
           1       0.47      0.30      0.36      3044
           2       0.57      0.68      0.62      2964

    accuracy                           0.56      9000
   macro avg       0.55      0.57      0.55      9000
weighted avg       0.55      0.56      0.55      9000

Index(['iter', ' loss'], dtype='object')
```



LSTM Model

LSTM Model is in: LSTM.ipynb

In the LSTM model, I modified the padding.

Here is the first version of the make_word2vec_vector_lstm(sentence):

```
max_sen_len = top_data_df_small.stemmed_tokens.map(len).max()
padding_idx = w2vmodel.wv.key_to_index['pad']
def make_word2vec_vector_lstm(sentence):
    num_padding = max_sen_len - len(sentence)
    padded_sentence = [padding_idx] * num_padding +
[w2vmodel.wv.key_to_index.get(word, 0) for word in sentence]
    return torch.tensor(padded_sentence, dtype=torch.long,
device=device).view(1, -1)
```

It calculates the number of padding required for the sentence to match the max_sen_len. Then it creates a new list with the required number of padding indices in the sentence from word2vec model's vocabulary. If the word is not in the vocabulary, it is set to 0. This pads the sentence with the padding index to make it the same length as max_sen_len. It is added before the actual words in the sentence so the resulting tensor has a fixed length.

I also modified section 10 and 11 to better fit the LSTM model, for more detail, you can look at the code in those two sections.

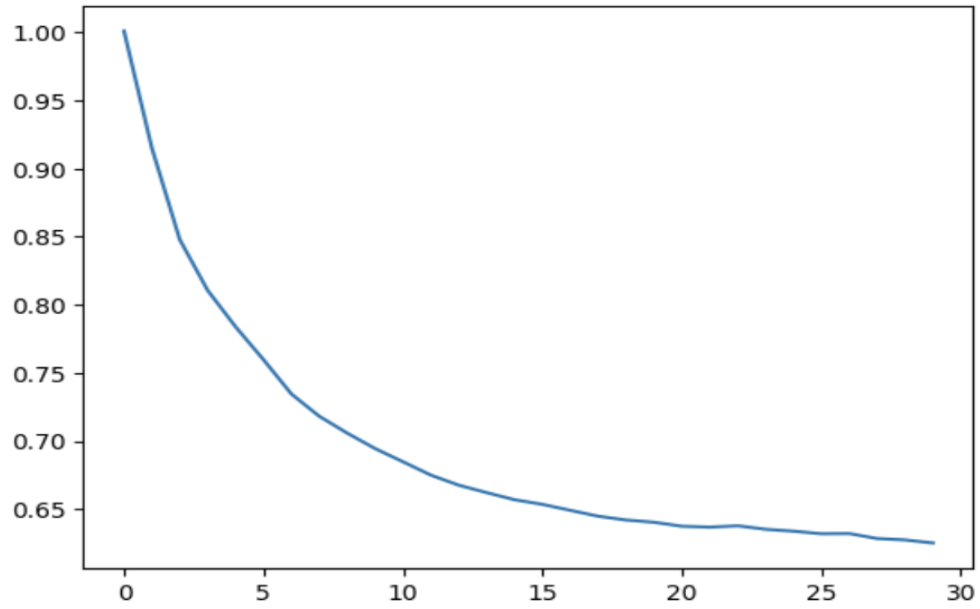
With above changes, I got following result with number of layers = 1, and hidden layer = 50:

```
Index(['iter', ' loss'], dtype='object')
              precision    recall  f1-score   support

           0       0.74      0.77      0.76      2992
           1       0.61      0.61      0.61      3044
           2       0.79      0.75      0.77      2964

    accuracy                           0.71      9000
   macro avg       0.71      0.71      0.71      9000
weighted avg       0.71      0.71      0.71      9000

Index(['iter', ' loss'], dtype='object')
```



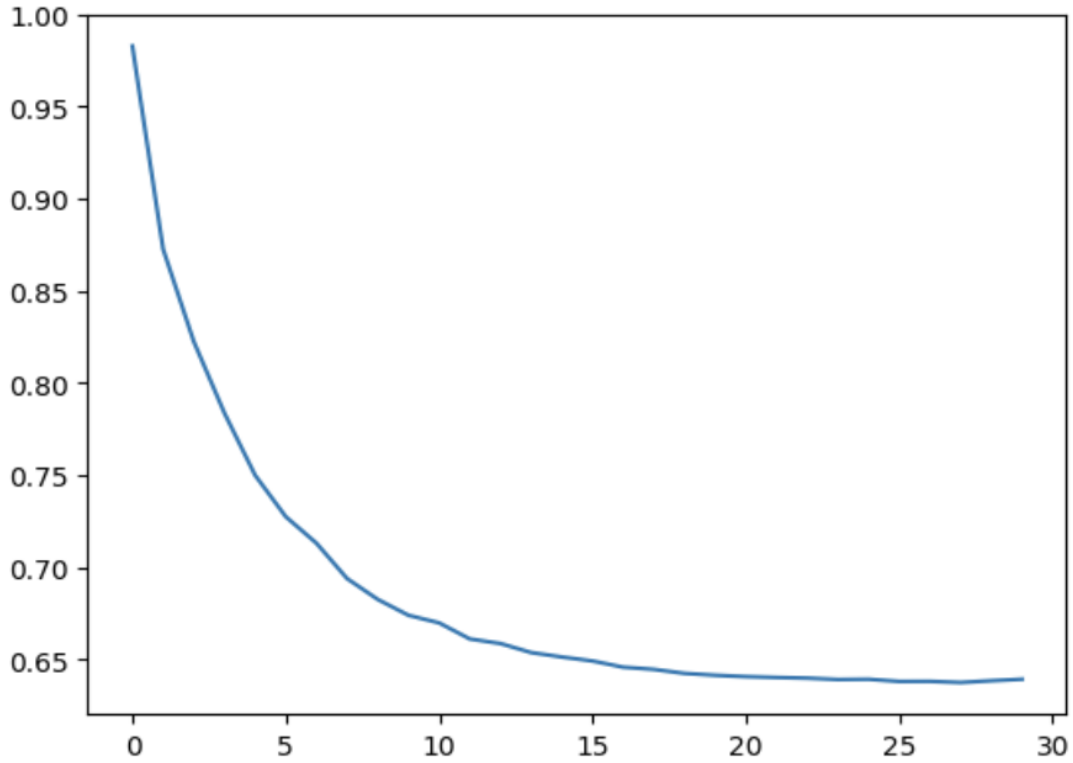
I also try it with different hidden layers. With different numbers of layers and hidden layers, the training time might increase depending on the layers. So the LSTM model takes more time to train compared to the CNN model in general.
The following shows the result of the number of layer = 1 and hidden layer = 100, which does not improve the accuracy.

```
Index(['iter', ' loss'], dtype='object')
              precision    recall  f1-score   support

           0       0.76      0.73      0.74      2992
           1       0.61      0.59      0.60      3044
           2       0.75      0.80      0.77      2964

    accuracy                           0.71      9000
   macro avg       0.71      0.71      0.71      9000
weighted avg       0.71      0.71      0.71      9000

Index(['iter', ' loss'], dtype='object')
```



I went back to modify the make_word2vec_vector_lstm function to see if I can have better accuracy. Here is the final version of the function:

```
max_sen_len = top_data_df_small.stemmed_tokens.map(len).max()
padding_idx = w2vmodel.wv.key_to_index['pad']
def make_word2vec_vector_lstm(sentence):
   return torch.tensor([w2vmodel.wv.key_to_index.get(word, padding_idx)
for word in sentence], dtype=torch.long, device=device).view(1, -1)
```

In this version, padding is not explicitly added to the sentence.
I tried it with different parameters, and the following is the best result, the number of layers = 1, and hidden layer = 100, I got 72% accuracy which is a little better than the previous version.

```
Index(['iter', ' loss'], dtype='object')
              precision    recall  f1-score   support

           0       0.76      0.75      0.76      2992
           1       0.63      0.61      0.62      3044
           2       0.76      0.80      0.78      2964

    accuracy                           0.72      9000
   macro avg       0.72      0.72      0.72      9000
weighted avg       0.72      0.72      0.72      9000
```

```
Index(['iter', ' loss'], dtype='object')
```