

IEMS5726 Data Science in Practice(Fall 2022)

Course Project

Student Name: Li Kaixu

Student ID:1155180259

Project Number: 4

Project Title: Semantic Analysis on Movie Review using RNN

(1) Problem definition:

Definition:

Semantic analysis means analyzing the emotion or feelings from a piece of text to get the “meaning” of text. In this particular problem, this means getting the feeling of viewers to a movie from the text of his comment. The result will be divided into “positive” and “negative”.

Why important:

- [1] help us know better about the feedback of views of a particular movie or other
- [2] help us know better about the preferences of a particular customer and adjust the recommendation system to offer better service
- [3] can be extended to other fields, like analyzing feedback online purchasing or individual-customed recommendation list

(2) Data source:

The dataset is “IMDB Dataset of 50K Movie Reviews”. This can be viewed as lines of data in format <comment text>:<feeling> pairs. The comment text is a text of movie review(multiple lines). The feeling part is simply positive/negative.

Link to dataset: <https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews>

(3) Data preprocessing:

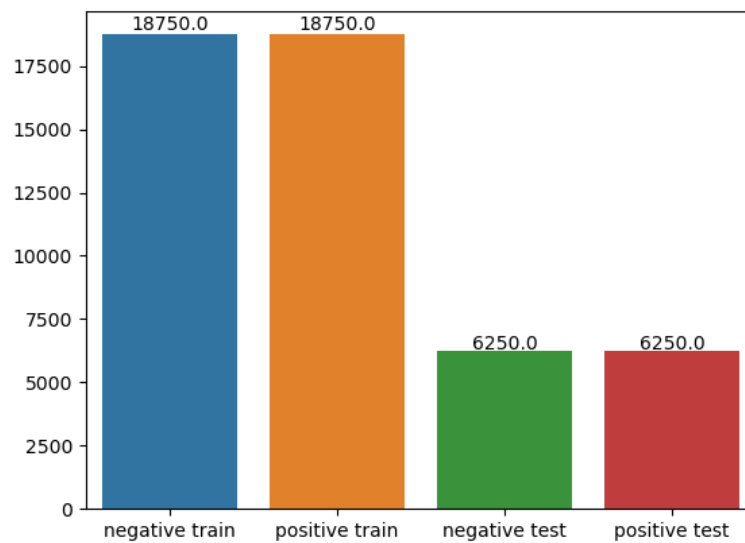
The following steps are used to preprocess data:

- [1] load IMDB Dataset text file as Pandas Dataframe, the column “review” as X(feature) and column ‘sentiment’ as y(label)

- [2] split training and testing data(in practice, 37500 for training, 12500 for testing)
- [3] tokenization the 'review data', build the vocabulary on the input, converting the original review text to a list of tokens
- [4] define a map on vocabulary to [1,...,n] based on the occurrence frequency, replace the tokens with its mapping number
- [5] padding the review list to length 500 with 0 before the comment
- [6] convert the label y positive/negative to 1/0

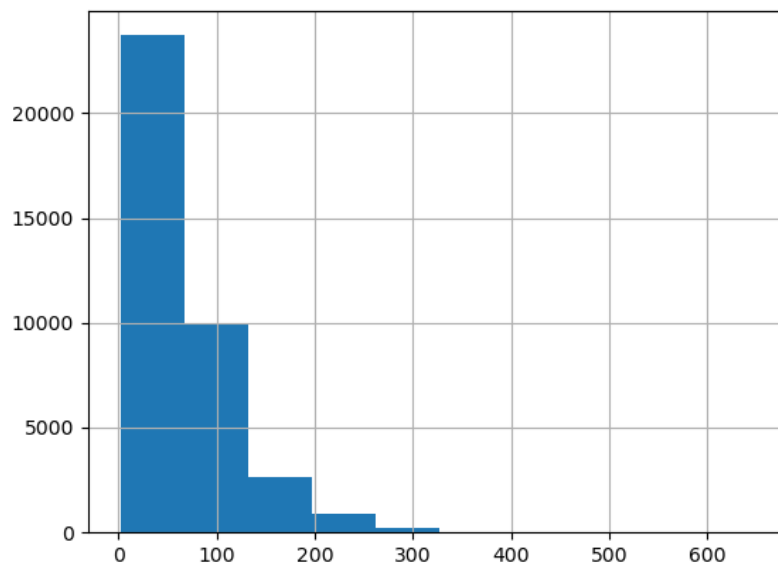
Visualization:

- [1] figure of training-testing y distribution:



The figure will be in "2-distribution_tot.png" when running the code.

- [2] distribution of the review length



[3] The result of input X and Y for Pytorch tensor:(this is not suitable to build figure)

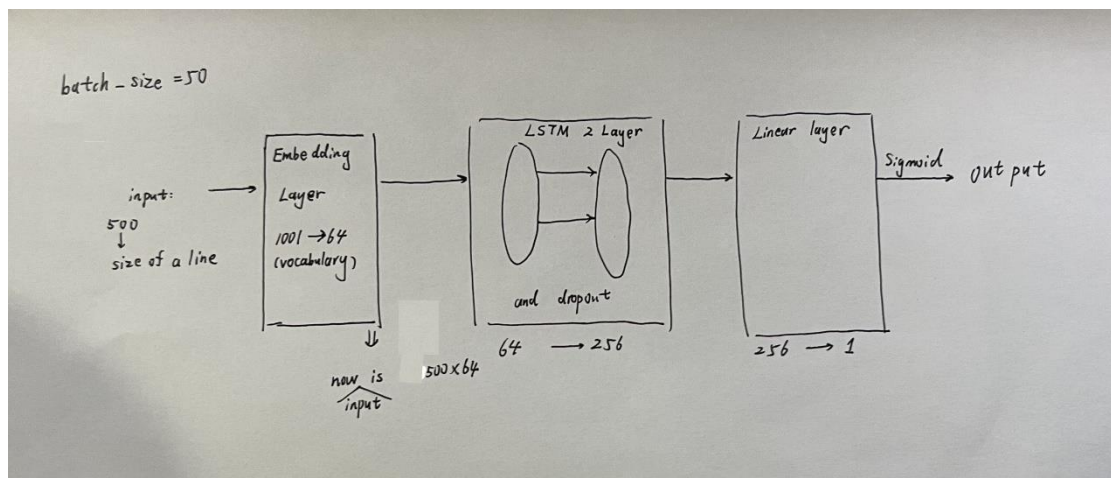
```
Sample input size: torch.Size([50, 500])
Sample input:
tensor([[ 0,  0,  0, ..., 345,  48, 807],
        [ 0,  0,  0, ...,  4, 334,  9],
        [ 0,  0,  0, ...,  4, 314, 588],
        ...,
        [ 0,  0,  0, ...,  8,  5, 29],
        [ 0,  0,  0, ..., 679, 88,  1],
        [ 0,  0,  0, ..., 137, 22, 368]], dtype=torch.int32)
Sample input:
tensor([0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1,
        0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0,
        1, 0], dtype=torch.int32)
```

(4) Data analysis:

We use Machine Learning method, build a simple LSTM network and train the parameters. The layers are defined as following:

```
SentimentRNN(
  (embedding): Embedding(1001, 64)
  (lstm): LSTM(64, 256, num_layers=2, batch_first=True)
  (dropout): Dropout(p=0.3, inplace=False)
  (fc): Linear(in_features=256, out_features=1, bias=True)
  (sig): Sigmoid()
)
```

We can know the structure of network is as following:



The output is a probability of being positive, naming pro. Then the probability of being negative is (1 - pro).

According to optimizing, we use nn.BCELoss() as training loss and Adaboost.

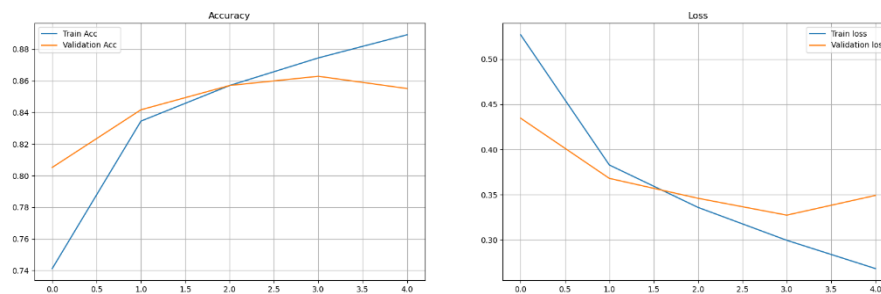
Metrics for judging:

According to judgement, we use both Binary Cross Entropy Loss (nn.BCELoss()) and accuracy.

Visualization:

The following figure described the trend of training/testing accuracy and loss Binary

Cross Entropy Loss according to number of epoch (3-acc.png when running)



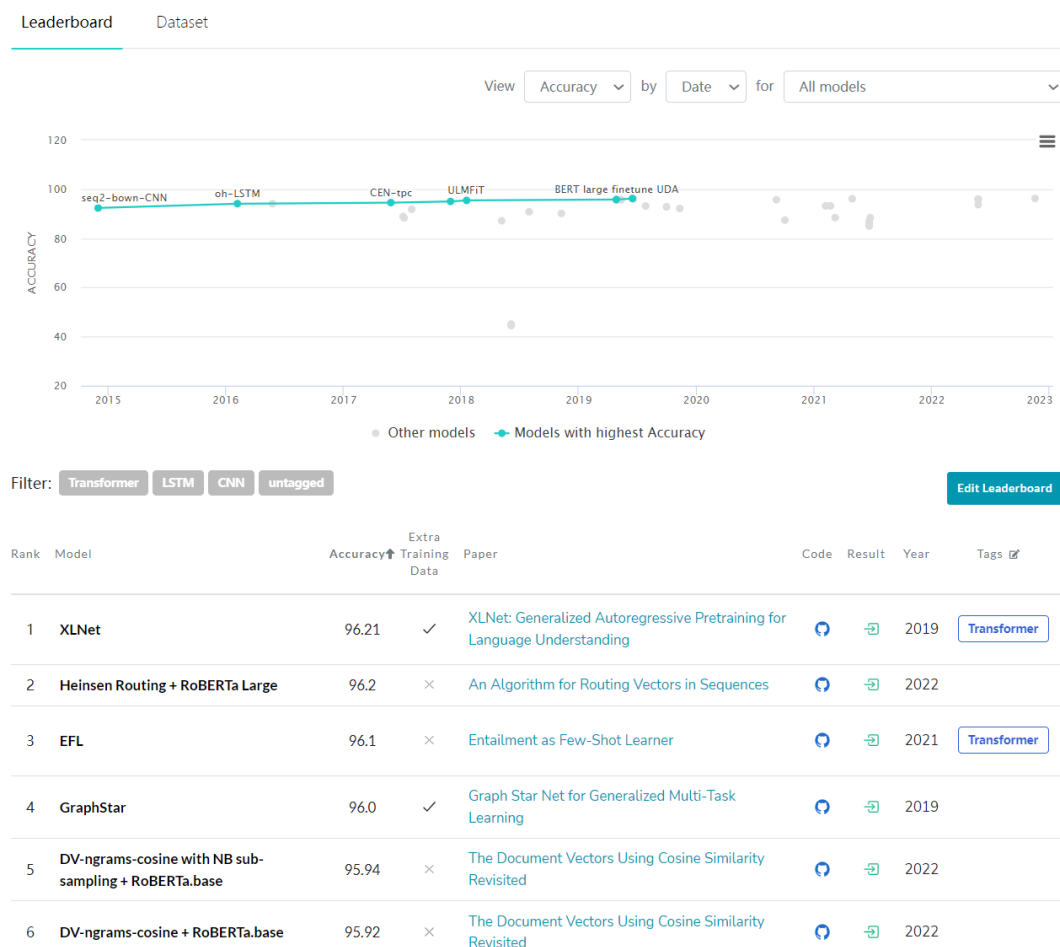
It is each to see most of the time both accuracy grows and the loss decrease, However, in about epoch 4-5, the accuracy of testing actually decreases and the loss sees a grow. This means for this model, we need to carefully decide which model we choose, too much training may do the negative effect.

About the exact numbers, we have to admit that 88% accuracy is not so high compared with top ones in public, this is a link to show the results of other methods

(<https://paperswithcode.com/sota/sentiment-analysis-on-imdb>):

😊 Sentiment Analysis

Sentiment Analysis on IMDB



It is easy to know many other methods are above 95% accuracy, so it still needs to be

promoted. Although 88% accuracy seems able to be used, but it is quite low compared to other public methods.

(5) Conclusion and discussions:

In the reproduction, we preprocess the raw text data for the input of machine learning method. Then, we can build a simple 2-layer LSTM network to analyze the feelings of comments. We get a good result and prove it make sense to train a machine learning model on this IMDB dataset.

Simply viewing the accuracy of 88% is not bad. However, compared to other popular methods, it still needs much promotion to increase accuracy. Also, we still need to apply it on more data to evaluate its performance.

(6) Extension of course project in part2

[1] Item2: Use pre-trained word embeddings like GloVe word embeddings

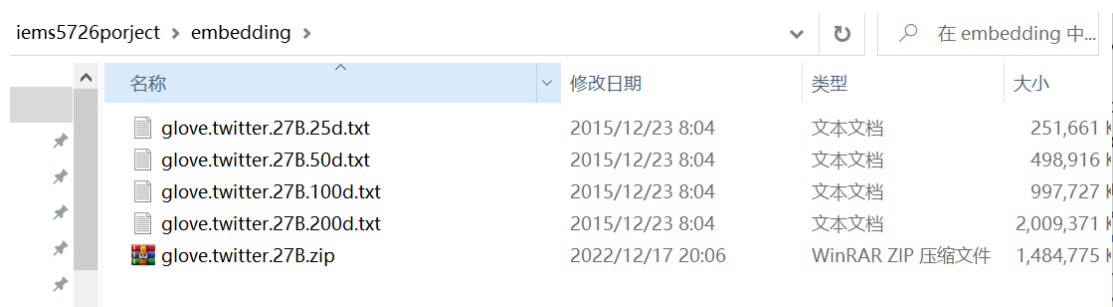
In this part, we choose GloVe word embedding to replace the original embedding part. The official link of Glove on GitHub is <https://github.com/stanfordnlp/GloVe> . On GitHub we can see there are following embeddings we can use:

Download pre-trained word vectors

The links below contain word vectors obtained from the respective corpora. If you want word vectors trained on massive web datasets, you need only download one of these text files! Pre-trained word vectors are made available under the [Public Domain Dedication and License](#).

- Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): [glove.42B.300d.zip](#) [mirror]
- Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): [glove.840B.300d.zip](#) [mirror]
- Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 300d vectors, 822 MB download): [glove.6B.zip](#) [mirror]
- Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 200d vectors, 1.42 GB download): [glove.twitter.27B.zip](#) [mirror]

In our compliment, we use the last one, glove twitter 27B embedding. When unpacked we can see four different txt documents:



名称	修改日期	类型	大小
glove.twitter.27B.25d.txt	2015/12/23 8:04	文本文档	251,661 KB
glove.twitter.27B.50d.txt	2015/12/23 8:04	文本文档	498,916 KB
glove.twitter.27B.100d.txt	2015/12/23 8:04	文本文档	997,727 KB
glove.twitter.27B.200d.txt	2015/12/23 8:04	文本文档	2,009,371 KB
glove.twitter.27B.zip	2022/12/17 20:06	WinRAR ZIP 压缩文件	1,484,775 KB

We use the first one. This means we choose the glove twitter embedding with 25 dimensions. We convert it into dictionary and directly use it to change the input. Look back what we use in original input:

```

Sample input size: torch.Size([50, 500])
Sample input:
tensor([[ 0,  0,  0, ..., 345,  48, 807],
        [ 0,  0,  0, ...,  4, 334,  9],
        [ 0,  0,  0, ...,  4, 314, 588],
        ...,
        [ 0,  0,  0, ...,  8,  5, 29],
        [ 0,  0,  0, ..., 679, 88,  1],
        [ 0,  0,  0, ..., 137, 22, 368]], dtype=torch.int32)
Sample input:
tensor([[0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1,
        0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0,
        1, 0], dtype=torch.int32)

```

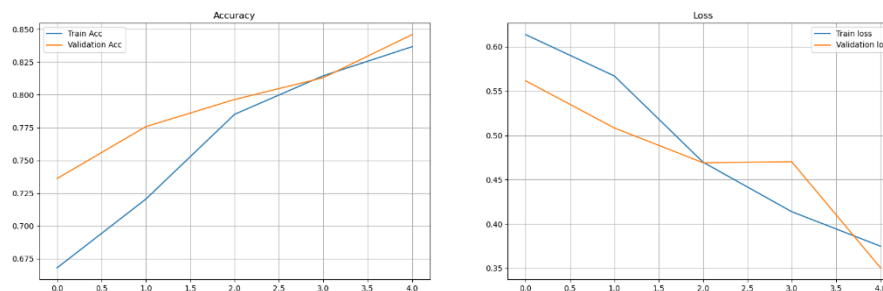
Since we directly use 25d embedding, the shape will become $50 \times 500 \times 25$, each number in X will be replaced by the word embedding of it (a 25-d float vector). And the RNN will become:

```

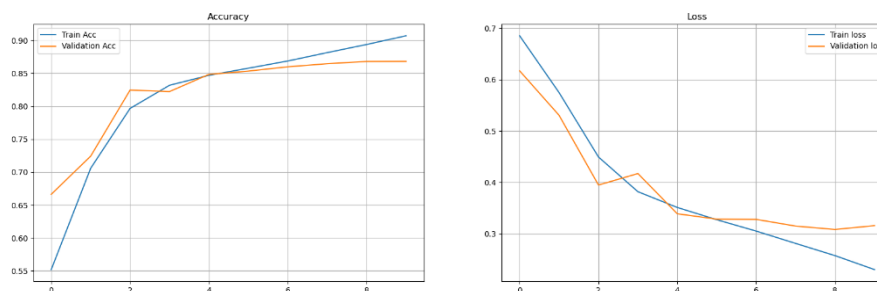
SentimentRNN(
  (lstm): LSTM(25, 256, num_layers=2, batch_first=True)
  (dropout): Dropout(p=0.3, inplace=False)
  (fc): Linear(in_features=256, out_features=1, bias=True)
  (sig): Sigmoid()
)

```

Then we see the output accuracy and loss:



We can see under the same number of epochs of 5, the accuracy is slightly lower than previous one. However, considering that the tendency, it does not really converge. We manually change the epoch to 10 and see it.

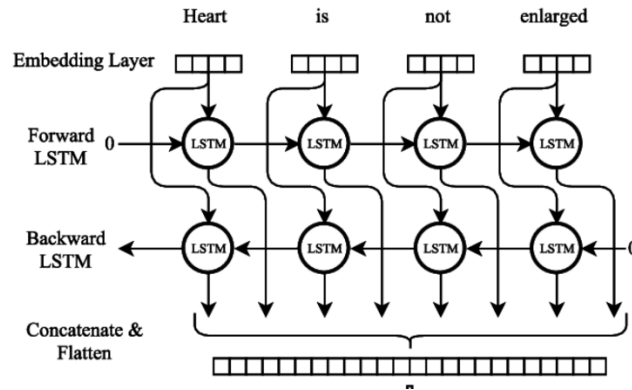


We can see the training accuracy can be up to 90%, while testing accuracy still 86%. (epoch=10 in commented in real code, we still submit the version of epoch = 5 in it). The performance is slightly better than original one.

[2] Item4: Increase the model complexity by using bidirectional LSTMs

Bidirectional LSTM, or bi-LSTM, is a model that consists of two LSTMs: one taking the input in a forward direction, and the other in a backwards direction. In

<https://paperswithcode.com/method/bilstm> there is a brief picture to show it:



In pytorch nn, it is rather simple to complete that. See the document on <https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html>, we can know that we simply add parameter 'bidirectional=True'. What we need to pay attention is that a bidirectional LSTM has 2 layers. So the hidden layer connected to the linear part(fc in our code) should be $2 \times \text{hidden node number}$. The training parameter should be $(\text{no_layers} \times 2) \times \text{batch_size} \times \text{hidden_dim}$. There are other few places need to change the dimension.

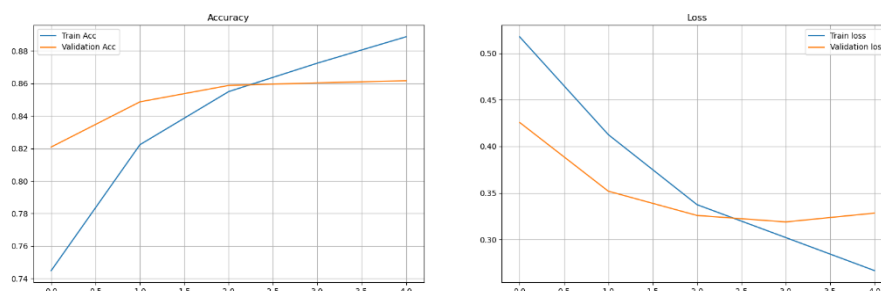
We first show our model and input format. The figure is in following page. We can see the input format is as the original one. The difference is LSTM layer from 2 LSTM to 2 bidirectional LSTM layer(this means we have 4 LSTM layers, 2 forward and 2 backward).

```
Sample input:
tensor([[ 0,  0,  0, ..., 507,  3, 40],
        [ 0,  0,  0, ..., 35,  1, 659],
        [ 0,  0,  0, ..., 251, 491,  1],
        ...,
        [ 0,  0,  0, ..., 15, 189, 124],
        [ 0,  0,  0, ..., 147, 145, 627],
        [ 0,  0,  0, ..., 28, 579,  60]], dtype=torch.int32)

Sample input:
tensor([0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0,
        1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0,
        1, 0], dtype=torch.int32)

SentimentRNN(
  (embedding): Embedding(1001, 64)
  (lstm): LSTM(64, 256, num_layers=2, batch_first=True, bidirectional=True)
  (dropout): Dropout(p=0.3, inplace=False)
  (fc): Linear(in_features=512, out_features=1, bias=True)
  (sig): Sigmoid()
)
```

The training result is following, we still use the same loss function and accuracy to evaluate our model:



From the figure we can know the accuracy and loss does not really see a improvement.

We need to combine other methods to see if it really make a difference.

(7) About the structure of submitted project folder

The folder have the following files:

- IEMS5726_report_1155180259.pdf : report
- IEMS5726_ProjectPart1_1155180259.py : reproduction
- IEMS5726_ProjectPart2Item2_1155180259.py : item2 with Glove embedding
- IEMS5726_ProjectPart2Item4_1155180259.py item4 with bidirectional LSTM
- IMDB Dataset.csv : Input training dataset, must exist in the root folder
- glove.twitter.27B.25d.txt: exported word embedding for items 2, must exist in the root folder

After running IEMS5726_ProjectPart1_1155180259.py, new files will be:

- 1-distribution.png distribution of training y positive and negative
- 2-distribution_tot.png distribution of both training and testing y
- 2-text_length.png distribution of training text length
- 3-acc.png the accuracy and loss plot based on number of epoch
- vocab.txt a mapping dictionary from the sorted vocabulary to list[1,...,n]
- state_dict.pt saved model of sentiment network RNN, this is the one after the final epoch

After running IEMS5726_ProjectPart2Item2_1155180259.py(the embedding txt is needed), new files will be:

- 3-acc2_gloveEmbedding.png the accuracy and loss plot based on number of epoch
- state_dict_gloveEmbedding.pt saved model of sentiment network RNN, this is the one after the final epoch

After running IEMS5726_ProjectPart2Item4_1155180259.py(the embedding txt is needed), new files will be:

- 1-distribution3_bi.png distribution of training y positive and negative
- 2-text_length3_bi.png distribution of training text length
- 3-acc3_bi.png the accuracy and loss plot based on number of epoch
- vocab.txt a mapping dictionary from the sorted vocabulary to list[1,...,n]
- state_dict_bilayer.pt saved model of sentiment network RNN, this is the one after the final epoch