Your Name : Changhyun, Lee

Your Andrew ID : changhyl

Homework 3

1. Training Set Construction (5 pts)

Construct the training set for the amazon review dataset as instructed and report the following statistics.

Statistics	
the total number of unique words in T	21959
the total number of training examples in T	2000
the ratio of positive examples to negative examples in T	1
the average length of document in T	187.7695
the max length of document in T	3817

2. Performance of deep neural network for classification (20 pts)

Suggested hyperparameters:

1. Data processing:
    a. Word embedding dimension: 100
    b. Word Index: keep the most frequent 10k words
2. CNN
    a. Network: Word embedding lookup layer -> 1D CNN layer -> fully connected layer -> output prediction
    b. Number of filters: 100
    c. Filter length: 3
    d. CNN Activation: Relu

e. Fully connected layer dimension 100, activation: None (i.e. this layer is linear)

3. RNN:
    a. Network: Word embedding lookup layer -> LSTM layer -> fully connected layer(on the hidden state of the last LSTM cell) -> output prediction
    b. Hidden dimension for LSTM cell: 100
    c. Activation for LSTM cell: tanh
    d. Fully connected layer dimension 100, activation: None (i.e. this layer is linear)

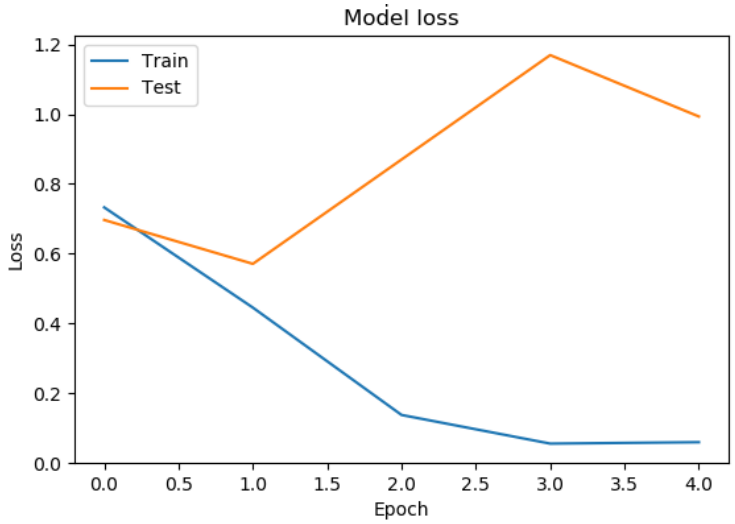
	Accuracy	Training time(in seconds)
RNN w/o pretrained embedding	76.45%	3104.21
RNN w/ pretrained embedding	74.25%	3148.34
CNN w/o pretrained embedding	81.60%	241.57
CNN w/ pretrained embedding	80.45%	198.98

3. Training behavior (10 pts)

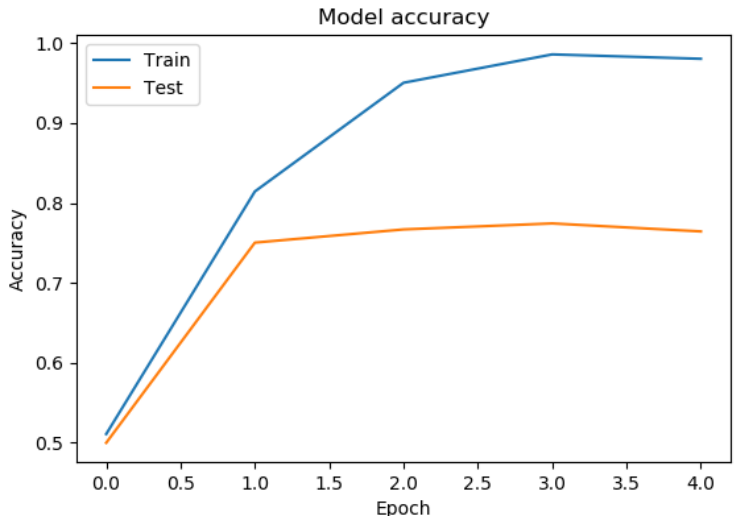
Plot the training/testing objective, training/testing accuracy over time for the 4 model combinations (correspond to 4 rows in the above table). In other word, there should be 2*4=8 graphs in total, each of which contains two curves (training and testing).

RNN w/o pretrained embedding

- training/testing objective over time

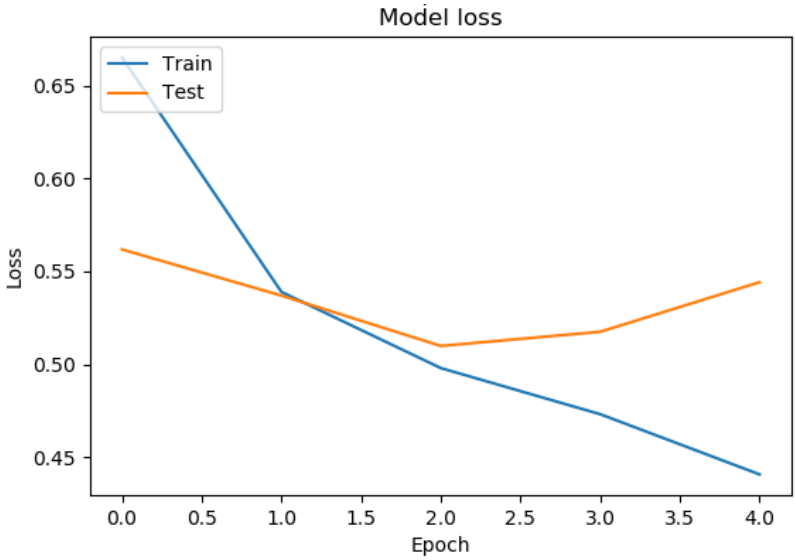


- training/testing accuracy over time

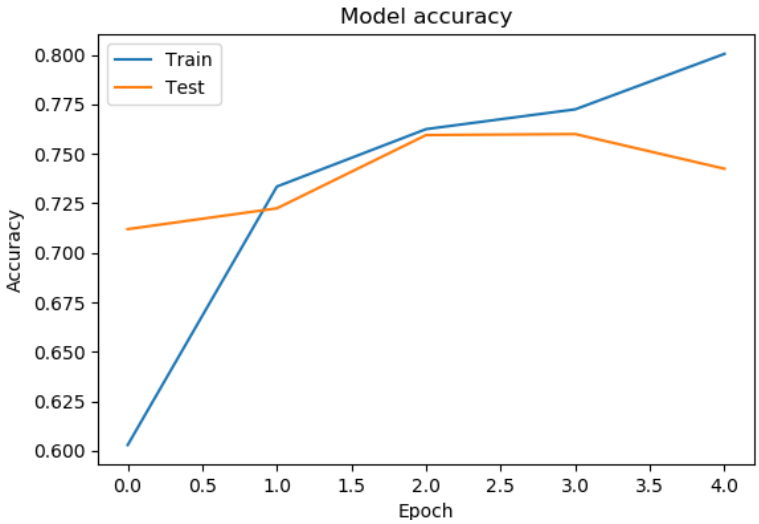


RNN w/ pretrained embedding

- training/testing objective over time

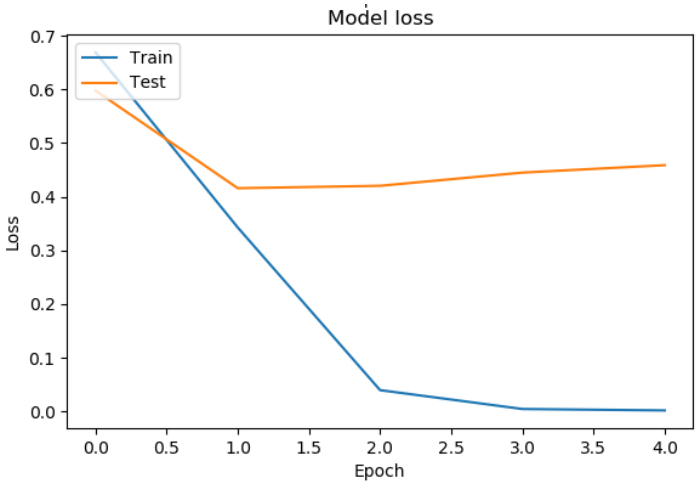


- training/testing accuracy over time

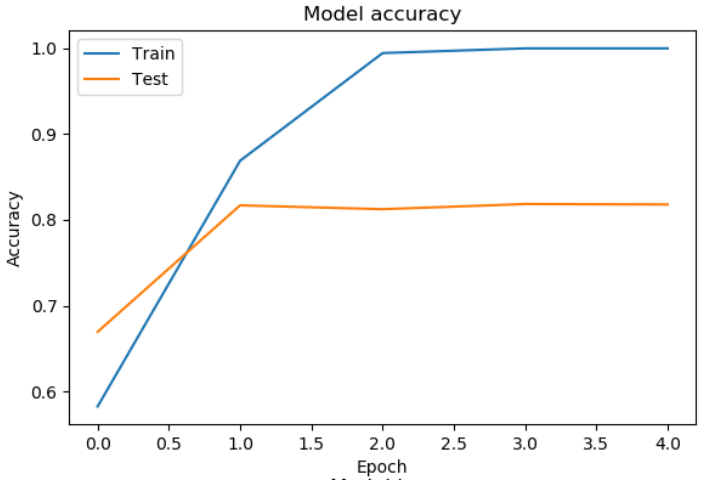


CNN w/o pretrained embedding

- training/testing objective over time

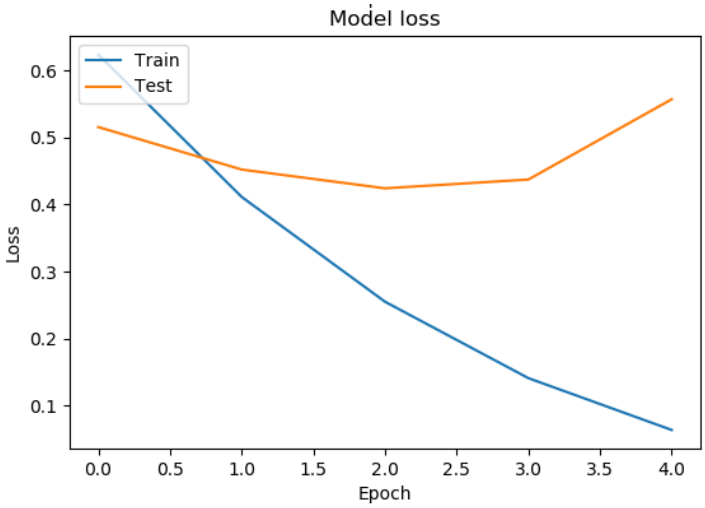


- training/testing accuracy over time

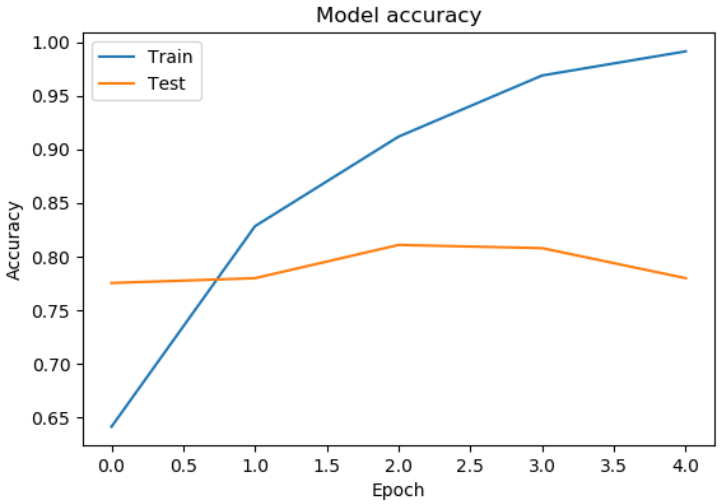


CNN w/ pretrained embedding

- training/testing objective over time



- training/testing accuracy over time



4. Analysis of results (10 pts)

Discuss the complete set of experimental results, comparing the algorithms to each other. Discuss your observations about the various algorithms, i.e., differences in how they performed, different parameters, what worked well and didn't, patterns/trends you observed across the set of experiments, etc. Try to explain why certain algorithms or approaches behaved the way they did.

- CNN vs LSTM

The results of CNN showed better performance than LSTM, in case of both with/without pretrained embedding. The important thing that we have to focus on is the training accuracy of CNN model reached about 100%, which means the starting point of overfitting. In case of computation time, the LSTM spent 15 times more than CNN. That is mostly because RNN based models have larger number of parameters.

As I do experiment, the LSTM tends to be trained slower than CNN model. It means, the value of loss converged slower in LSTM than CNN model. Although I did not spend more time to tuning parameters, if I tune some of the parameters of LSTM model, I can guess it because the CNN performance of CNN model looks like saturated, starting with overfitting to training data.

- With pre-trained embedding vs without pre-trained embedding

  Actually, I expected that the pre-trained embedding vectors can improve the performance of text classification. However, in both CNN and LSTM model, the pre-trained text classification does not work well in this task. It is mostly because the pretrained embedding vectors which is from GloVe dataset does not optimized in this binary text classification. From this experience, I could learn that I have to consider what tasks do we apply. The more fit the task I implement with GloVe's sentiment thermometer task, the better performance can be shown.

5. The software implementation (5 pts)

Add detailed descriptions about software implementation & data preprocessing, including:

1. A description of what you did to preprocess the dataset to make your implementations easier or more efficient.

I tried to concentrate the data generation, because the preprocessing heavily affect performance of classification. Each preprocessing technique is described as follows.

- Shuffle : We used data shuffle option, to prevent the model from overfitting.
- Padding : Each embedded word is padded by 0, to make the vector length same.
- Global Maxpooling : If we get output from the last hidden state, we will stuck at 50% accuracy, because the output is derived from just padded word which does not include

any meaning. Therefore, we used Global Maxpooling, with deriving the output from maximum value of each hidden states.

- Remove stopwords : we only extracted the characters of dataset with a-z or A-Z, and additionally removed stopwords that is in nltk.corpus.stopwords library, which is implemented by ‘w for w in words if not w in stopwords’.

2. A description of major data structures (if any); any programming tools or libraries that you used;

Computer specification :

- CPU : Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz
- RAM : 8GB

Programming tools :

- Tensorflow(Keras),
- Python 3.6

Library :

- Numpy
- Pandas
- Re
- Matplotlib
- Time
- Glob
- Stopwords
- Keras.preprocessing
- Keras.models
- Keras. layers

Functions :

- Get_sentence_list : get sequences from our data
- Tokenizer.texts_to_sequences : get index of our sequences from vocaburary.
- Pad_sequences : implement padding, to make the length of sequences same.
- Get_pre_trained_embedding : make embedding_matrix to get pretrained embedding matrix.
- Get_model_cnn : automatically generate cnn model
- Get_model_cnn_pre_trained : automatically generate cnn model which includes pretrained embedding layer.
- Get_model_lstm : automatically generate lstm model
- Get_model_lstm_pre_trained : automatically generate lstm model which includes pretrained embedding layer.

- Visualize : visualize the result of loss and acc, and show the plt.

3. Strengths and weaknesses of your design, and any problems that your system encountered;

Strength : Modularization. All of my codes are based on the functions described above. Therefore, it is very easy to regenerate similar codes. For this reason, both CNN.py and LSTM.py are implemented based on the modularized functions. Furthermore, it is easy to read how it can be processed.

Weakness : It does not include any class. If I used class to make overall system. I could make simpler my code and make easy to put member variables or methods.

Problem : I had a problem of LSTM with recording always 50% accuracy. 50% accuracy in binary classification (0:negative, 1:postivie) means that just randomly matched results. I noticed the reason, that is because I tried to find output in the last hidden state. Finally, this problem is solved by using Global maxpooling in the sequence of output values.