Your Name **ZihaoZHOU**

Your Andrew ID **zihaozho**

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	24473
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	187.677
the max length of document in T	3816

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	0.776999	205.857
RNN w/ pretrained embedding	0.861499	204.744
CNN w/o pretrained embedding	0.776499	36.062
CNN w/ pretrained embedding	0.831499	35.065

For speeding up the calculation, I use an AWS g4dn_xlarge(NVIDIA T4 GPU) instance to train the model. To control other variables to compare the used time, all epoch is set to 50. The training time includes the evaluation process so it may longer than actual training time.

PS: I keep all of the hyperparameters stated above but slightly change the structure of my CNN and RNN model. My structures are as follow:

CNN:

1. Word embedding lookup layer
2. Conv1d layer with Activation: Relu
3. **MaxPool1d with kernel size = 4 along**
4. **Dropout layer with dropout rate = 0.3**
5. Linear layer
6. Sigmoid output

LSTM:

1. Word embedding lookup layer
2. LSTM layer with Activation: Tanh
3. **MaxPool1d to extract reasonable hidden state as output**
4. **Dropout layer with dropout rate = 0.3**
5. Linear layer
6. Sigmoid output

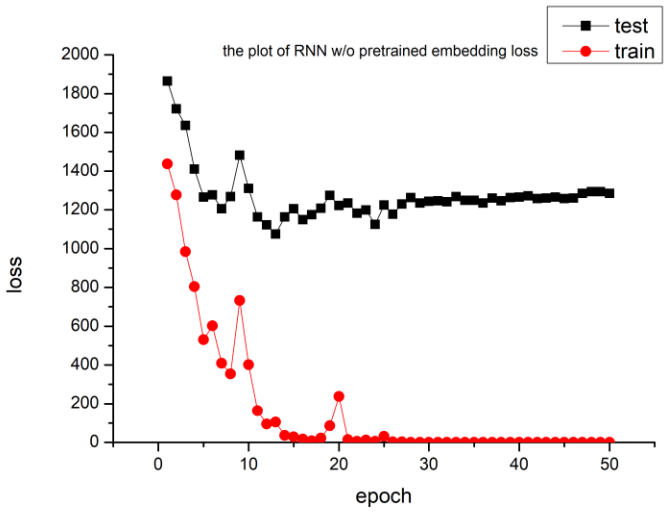
The reason for choosing this structure will be discussed in “4.Analysis of results”.

3. Training behavior (10 pts)

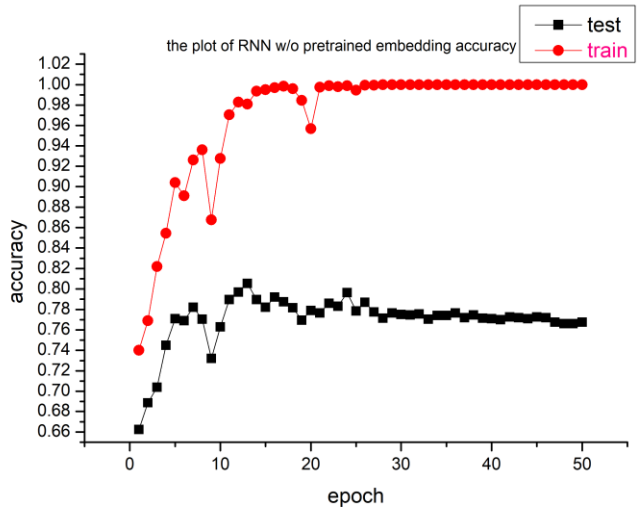
Plot the training/testing objective, training/testing accuracy over time for the 4 model combinations (correspond to 4 rows in the above table). In other words, there should be 2*4=8 graphs in total, each of which contains two curves (training and testing).

RNN w/o pretrained embedding

- training/testing objective over time

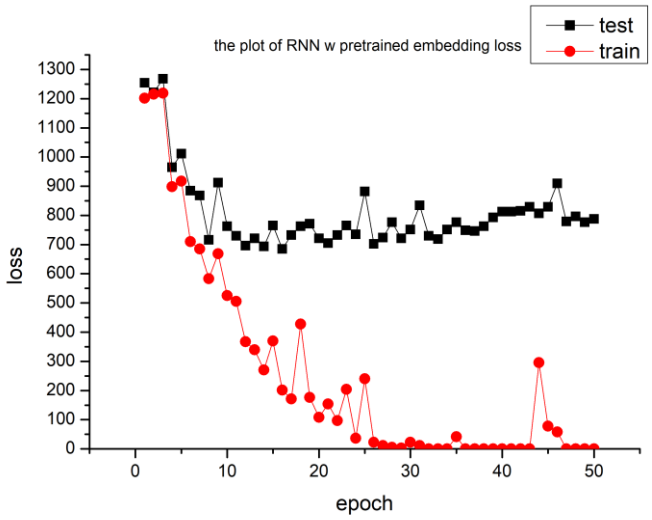


- training/testing accuracy over time

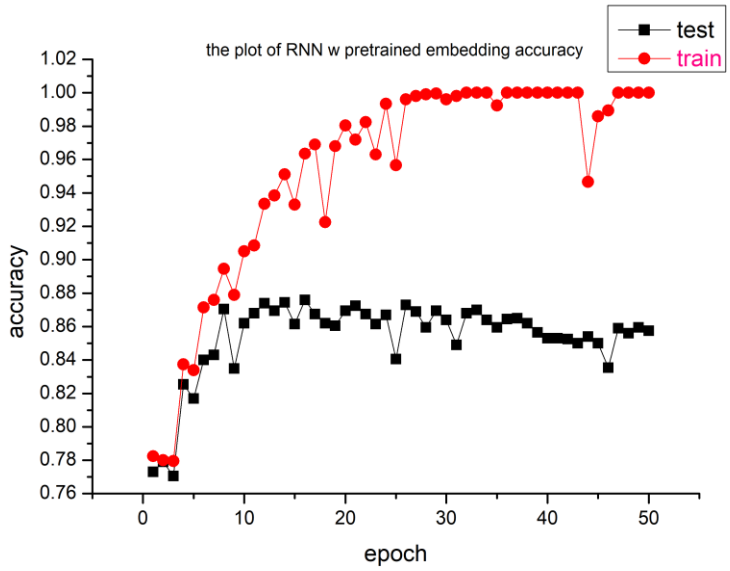


RNN w/ pretrained embedding

- training/testing objective over time

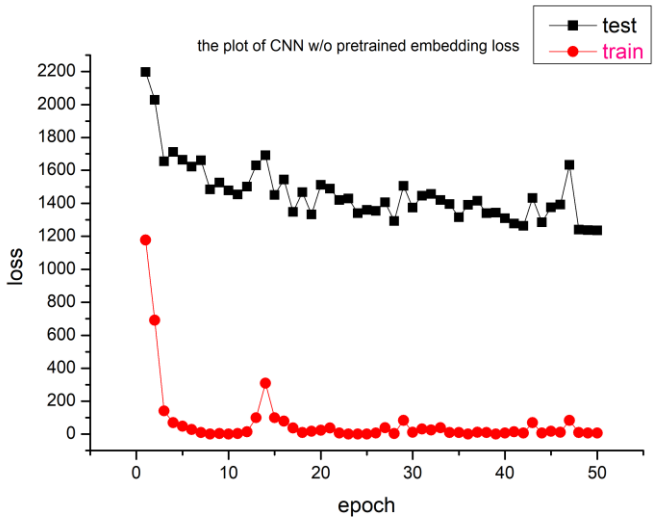


- training/testing accuracy over time

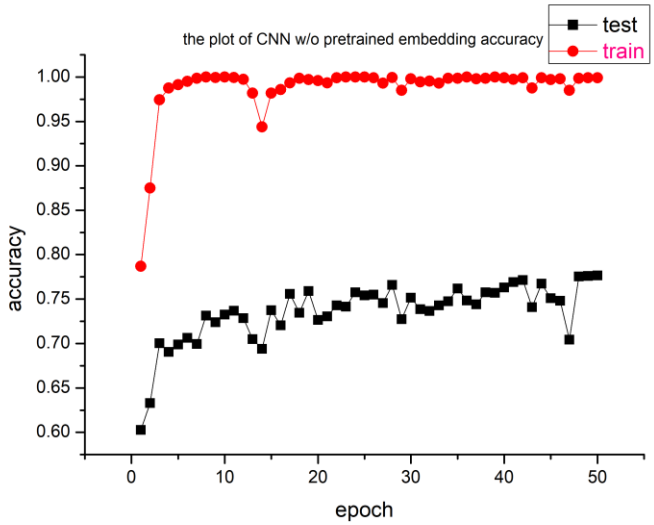


CNN w/o pretrained embedding

- training/testing objective over time

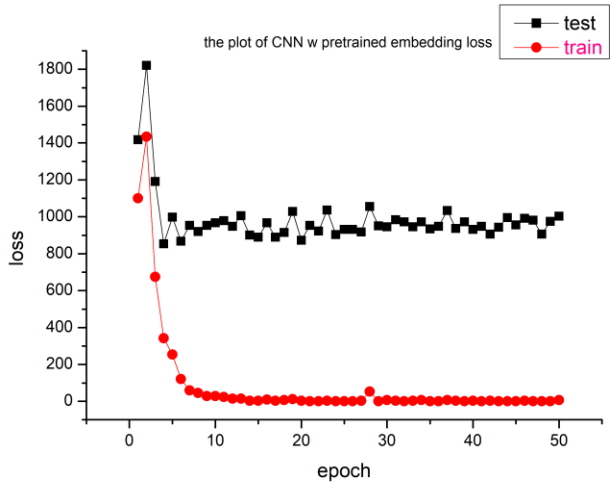


- training/testing accuracy over time

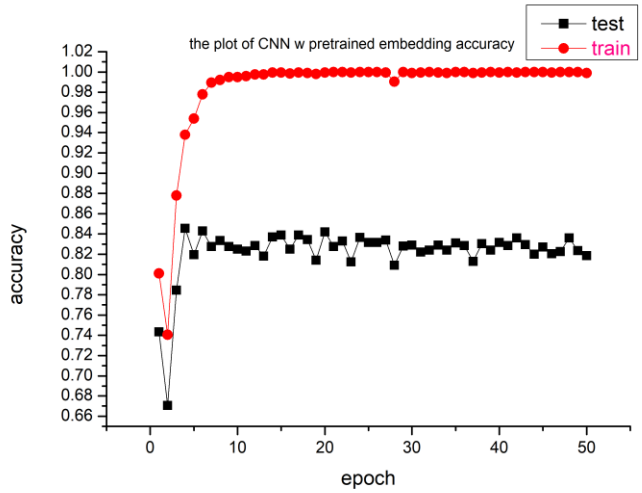


CNN w/ pretrained embedding

- 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.

1. Pretrained embedding or not

From both CNN and RNN experimental results, using pretrained embedding can dramatically increase the final test accuracy. The increment of accuracy for CNN is about 6% and RNN is 9%.

Using pretrained embedding can also slow the speed to converge. That is to say, from my observation, if not use pretrained embedding, after several epochs, it will reach the optimal(or local optimal) and shake there. If use pretrained embedding, it will take a few more epochs to reach the optimal for training accuracy or loss and at the same time, the accuracy continues increasing.

2. Using dropout layer or not

From the above plots, we can see that the performance gap between train and test is large. In some scenarios, even after using a dropout layer, train loss goes down to zero. This means the training data is not sufficient enough and one way to deal with this problem is to set a dropout layer. The improvement of using a dropout layer is about a 3% increment at accuracy.

3. Using MaxPool1d layer

For CNN, I use a MaxPool1d layer to extract the important element after the convolutional filter. This method increases by 5% accuracy.

For LSTM, according to the post in piazza from TA Zihang Dai, I use a max-pooling over the RNN states instead of taking the last hidden state because there are many '<pad>' tokens at the end of each sentence so it is unreasonable to pass hidden state over all of these paddings extract the lass hidden as output. This method dramatically increases accuracy from 63% to 86%.

4. CNN and RNN

These are two different models, from the final result. RNN performs slightly better than CNN. However, the time used by RNN is much longer than CNN because the transmission for the hidden state to the next LSTM core one by one can't be done parallelly.

5. The software implementation (5 pts)

Add detailed descriptions of 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 create a dictionary to hold each word in the training set and its corresponding number of occurrences. The top (10k -2) word will be selected and allocated an index from 2 to 10k-1

according to descending order. Index 0 and 1 are allocated to '<pad>' and '<unk>'. Then, every sentence in the training set will be converted to an index version and be fed into the neural network.

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

I use a class LSTMconfiguration() and class CNNconfiguration() to store the set config to initialize the neural network. Each class includes the necessary elements like embedding_size, filter_num, and maxpool_kernel_size etc.

The used library:

torch -- 1. build CNN and RNN 2. Dataloader to load train and test data

numpy-- to deal with array-like data

os -- to deal with path concatenation

time -- to record time

sys -- to receive argument

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

Strengths:

This program is a modularity type.

"CNN.py" and "LSTM.py" start execution.

"CNNwo.py", "CNNw.py", "RNNwo.py", "RNNw.py" implement four required training process corresponding to their name.

"pretrain_weight.py" extract pretrained embedding weight from the given file.

module.py" contains the CNN and LSTM module.

"util.py" contains the function that other programs should use.

Weakness:

I hardcoded the hyperparameters and the path to store the model in my program, to be tested by myself sufficiently. Additionally, with no input parameter, it is easier for TA to run it.