Your Name: Seowoo Han

Your Andrew ID: seowooh

Homework 2

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	22764
the total number of training examples in T	2000
the ratio of positive examples to negative examples in T	1.0
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)

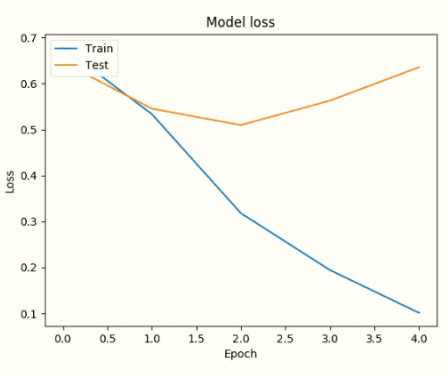
	Accuracy	Training time(in seconds)
RNN w/o pretrained embedding	0.7745	31.016
RNN w/ pretrained embedding	0.661	31.016
CNN w/o pretrained embedding	0.7915	1.0006
CNN w/ pretrained embedding	0.7165	2.0001

3. Training behavior (10 pts)

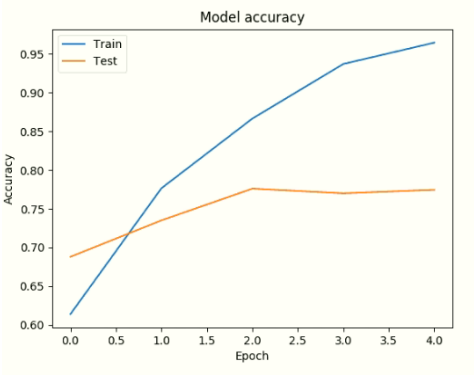
Plot the training/testing objective, training/testing accuracy over time for the 4 model combinations (correspond to 4 rows in the above table). In other word, there should be 2*4=8 graphs in total, each of which contains two curves (training and testing).

RNN w/o pretrained embedding

- training/testing objective over time

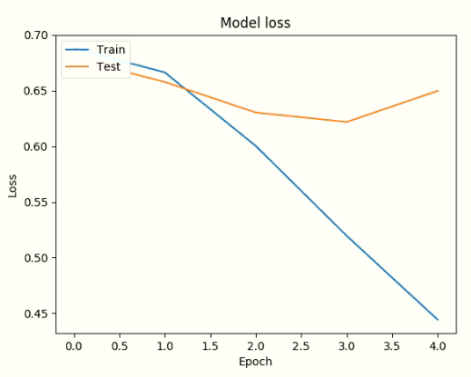


- training/testing accuracy over time

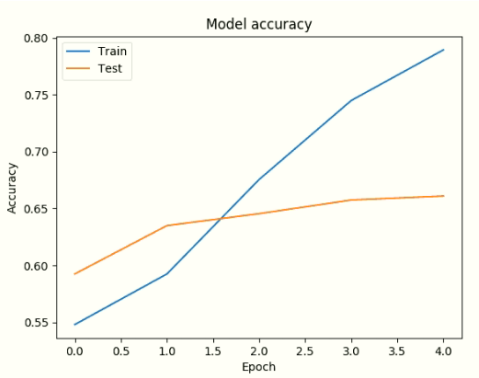


RNN w/ pretrained embedding

- training/testing objective over time

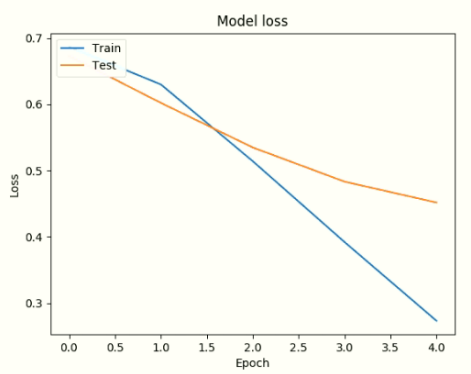


- training/testing accuracy over time

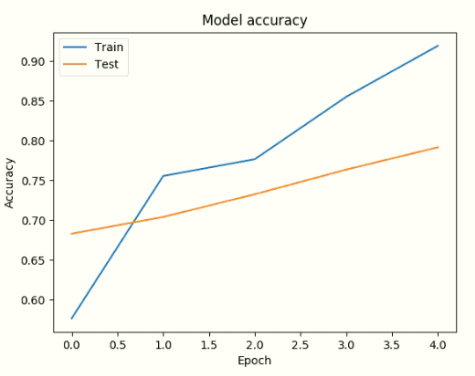


CNN w/o pretrained embedding

- training/testing objective over time

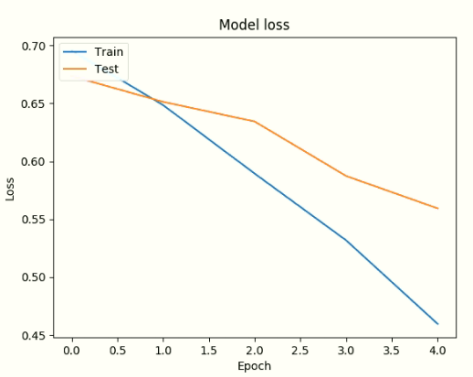


- training/testing accuracy over time

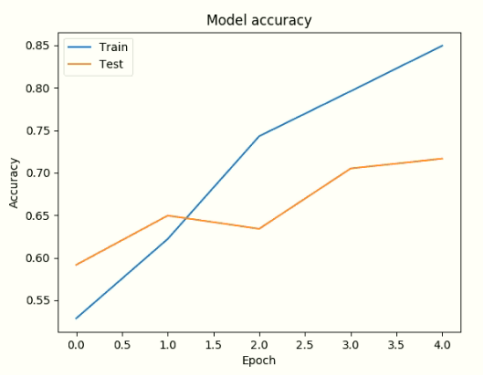


CNN w/ pretrained embedding

- 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. RNN

  RNN using LSTM took about 15 times more training time than the CNN algorithm. Because the hidden layer stores the result value of previously hidden layers, there is more data to process than CNNs, and processing time was much longer because data is processed sequentially.

  However, performance is better with the CNN, whether or not pre-trained embedding. This task was to classify whether a sentence was positive or negative. To find out if a sentence is positive or negative, look for specific words. (Positive: good, fine, like, love, negative: hate, not, unlike, etc.) Therefore, there is information that can be obtained only by the words themselves, not by considering the surrounding words and contexts.

  In theory, a similar condition should yield better results for the RNNs. Therefore, increasing the number of epochs in each network will increase accuracy and reduce the loss. In particular, the method of increasing the number of hidden dimensions for LSTM cells of the RNN can show optimal performance. Doing so can yield better performance than the CNN.

- Training model

  Some problems have arisen from learning with only the information in a given handout. The handout says not to use activation when setting up a fully connected layer. However, I used 'binarycrossentropy' as a loss function. When I made a linear output without using the activation function, I was trained, but when the test was conducted, the accuracy increased depending on what the test set used, and then converged to a value of 0. Therefore, after various attempts, I used the fully connected layer's activation function as a sigmoid, and the results were stable.

  I tested it by changing the batch size. The handout did not specify a batch size. So, when I set it to 32, which is the default value, the result came out well. To analyze the performance by changing the parameters, I changed the batch size to 16 and 64. When the activation function of the fully connected layer was not set, the accuracy increased and converged to a value of zero. On the contrary, when the activation function of the fully connected layer is set, the comparison is made between 32 and 64. At a batch size of 64, training and testing seemed to work well, but not on all networks. Batch size is also a parameter to consider for optimal performance. However, the code I wrote was limited to varying constraints and comparing the results. Therefore, I tested the batch size by setting it to 32, which shows consistently good results when tested multiple times.

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

Before this project, I thought that the result of pre-trained embedding would be good. However, the result was the opposite. If you look at the all.review.vec.txt file, there are many stop words at the top. In the train set I preprocessed, many stop words were eliminated, so it was not optimized for the pre-trained embedding matrix, so the network without pre-trained embedding showed better performance.

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.

- Data set

The data in the positive folder and the data in the negative folder are composed of one pandas.dataframe. To distinguish between positive and negative, I created a row called label in the dataframe and assigned a value of 1 for positive and 0 for negative. For efficient learning, I trained by shuffle positive and negative words together.

text	label
The	**0**
⋮	⋮
Good	**1**
Amazon	**0**
and	**0**

<The compose of data set(train_DF/test_DF) >

- Data preprocessing

Functions such as Tokenizer, fit_on_texts, word_index, and word_count in Keras.preprocessing.text module is used. However, I found a bug using this module. In Tokenizer, I use num_words to specify how many words are used most often in a document, and filters remove unwanted words, so I use those parameters. However, after using the tokenizer function and using the fit_on_texts and word_index functions, I found that the values set in the Tokenizer are reset. When I searched on Google, there were a lot of questions people asked about this problem, and I found that the only way to fix the problem was to modify the Keras module directly. So to get rid of stop words, I used the

word_count function to get the frequency of each word. The results showed that the top 100 words stopped words commonly spoken in NLP. Therefore, to remove the stop words, only the 101st to 10101st data are used in the data descending word_count.

In addition, only 100 words were used for learning. The longest word in the data is about 3200. After setting maxlen in Karas.preprocessing.sequence.pad_sequences function to 3200 and learning, the running time of CNN was not very different from that of 100, but the running time of RNN was very long. Only 100 words are used for learning efficiency.

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

- Development environment: Ubuntu 18.04 64bits, GPU (RTX 2080Ti), CPU (Intel® Core™ i7-7700 @3.60GHz)
- Programming tools: Keras, python (3.6)
- Libraries: see the 'requirements.txt' in my code file

<File list>

main.py: This file imports TotalDatasetToList, preprocessing, rnn_lstm, and cnn modules to train RNN, CNN, and derive test results. You can train train_model in main.py and draw a picture of the accuracy and loss according to the epoch.

Preprocessing.py: embedding matrix. It divides sentences of data stored in pandas.dataframe into words, calculates word frequency and removes stop words.

cnn.py: This file creates the CNN model. The function used is different with and without the pre-trianed embedding matrix.

rnn_lstm.py: This file creates the RNN with LSTM model. The function used is different with and without the pre-trianed embedding matrix.

NegativeDatasetToList.py: Save the files in the negative data folder in a pandas.dataframe and set the label to 0.

PositiveDatasetToList.py: Save the files in the positive data folder in a pandas.dataframe and set the label to 1.

TotalDatasetToList.py: This file uses PositiveDatasetToList and NegativeDatasetToList modules and creates the positive and negative files into one dataframe using the variables from each module.

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

Strength: The crucial functions are modular, making them easier to use in other projects and making the code easier to read. Also, my code can remove stop words in a simpler way and reduce running time than people who write functions to remove stop words.

Weakness: It is hard to change the value by hard coding the value of word index. My code can set up CNN and RNN as a function, but it is unnecessarily long because it is separated.

Problem: When running the program, accuracy and loss is not constant. This occurs because the weights do not converge to a specific value because of the small amount of learning. In other words, the value changes depending on which test set is entered. Hyperparameter initial values and more training are required to show optimal performance.

Reference

1. Website

- ✓ https://fasttext.cc/docs/en/english-vectors.html
- ✓ https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/
- ✓ https://machinelearningmastery.com/prepare-text-data-deep-learning-keras/
- ✓ https://cloud.google.com/blog/products/gcp/intro-to-text-classification-with-keras-automatically-tagging-stack-overflow-posts
- ✓ https://machinelearningmastery.com/how-to-develop-a-word-level-neural-language-model-in-keras/
- ✓ https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/

2. Discussion members: Insoo Kim, Namkyu Kim, Sooah Lee