BME695DL/ECE695: Homework 7

Spring 2021
Due Date: Monday, April 26, 2021 (11:59pm)

Turn in your solutions via BrightSpace.

1 Introduction

This homework has the following goals:

1. To gain insights into the workings of Recurrent Neural Networks. These are neural networks with feedback. You need such networks for language modeling, sequence-to-sequence learning (as in automatic translation systems), time-series data prediction, etc.
2. To understand the rather severe shortcomings of the traditional one-hot vector approach for representing the words.
3. To use RNNs for the modeling of variable-length product reviews provided by Amazon for automatic classification of such reviews.

2 Getting Ready for This Homework

Before embarking on this homework, do the following:

1. Carefully review the latest version of the Week 13 slides on "Recurrent Neural Networks for Text Classification." Make sure you understand what is meant by the gating mechanism to address the problem of vanishing gradients that are caused by the long chains created by the feedback in a neural network.
2. Download and install the newly released Version 2.0.8 of DLStudio. Note that if you '*pip install*' it directly from the 'pypi.org' website, you

will not get the changes in the Examples directory of the distribution. Version 2.0.8 includes text processing using word embeddings. We will talk about word embeddings next Tuesday in class.

3. Download the text datasets from the main documentation page for DLStudio into the Examples directory of the distribution and execute the following command on the downloaded gzipped archive:

   ```
   tar xvf text_datasets_for_DLStudio.tar.gz
   ```

   This will create a subdirectory 'data' in the Examples directory and deposit all the datasets in it.

   If you followed the previous instructions, you will find the following datasets in the 'Examples/data/' directory:

   ```
   sentiment_dataset_train_400.tar.gz        (vocab_size = 64,350)
   sentiment_dataset_test_400.tar.gz

   sentiment_dataset_train_200.tar.gz        (vocab_size = 43,285)
   sentiment_dataset_test_200.tar.gz

   sentiment_dataset_train_40.tar.gz         (vocab_size = 17,001)
   sentiment_dataset_test_40.tar.gz

   sentiment_dataset_train_3.tar.gz          (vocab_size = 3,402)
   sentiment_dataset_test_3.tar.gz
   ```

   Regarding the naming convention used for the archives, a number such as 200 in the name of a dataset means that the dataset is a collection of the first 200 reviews from each of the positive-reviews and the negative-reviews files in the subdirectories for each of the 25 product categories. In other words, the dataset with the number 200 in its name contains a total of 400 reviews for each product category. All the reviews pooled together are randomized and divided in 80 : 20 ratio between the training and the testing datasets. The last dataset shown above is just for your convenience as you are debugging your code.

4. After you have installed the datasets, it's time to become familiar with some actual code for text classification. Do this by executing the following script in the `Examples` directory of the distribution:

   ```
   text_classification_with_TEXTnet.py
   ```

This script uses DLStudio's `TEXTnet` as the RNN. You can use `TEXTnet` to create a neural network with feedback but the network you get will have no protection against vanishing gradients. As supplied, the script will load in the following dataset:

```
sentiment_dataset_train_40.tar.gz          (vocab_size = 17,001)
sentiment_dataset_test_40.tar.gz
```

The network uses one-hot vector representations for the words. The large size of these vectors leads to a network with around 10 million learnable parameters.

Your results should be similar to those shown on Slide 36 of the Week13 slides. These dismal results are caused by a combination of the vanishing gradients that bedevils the neural networks with feedback. In what follows, I'll refer to this dataset as the "dataset-40". Here are the results you are likely to see:

```
Number of positive reviews tested: 200

Number of negative reviews tested: 195

Confusion Matrix:

                      predicted negative    predicted positive
true negative:                0%                   100%
true positive:                0%                   100%
```

5. Next, execute the following script that uses a GRU based gated recurrence for the RNN. The name of the network used from DLStudio is `GRUnet`:

   ```
   text_classification_with_GRU.py
   ```

   You will notice that, as supplied, it will load in the same dataset-40 as for the previous script.

   Unless you have a powerful GPU at your disposal, you will notice that this script will take inordinately long to train. You would want this script to run since a GRU is meant to mitigate the vanishing-gradients problem. However, the very large size of the one-hot vectors keeps the GRU from doing its job.

The two examples mentioned above were based on the dataset-40. Now imagine how much harder you would make it for the networks to learn if you tried them on dataset-200 or dataset-400 with their much larger vocabulary sets. **With dataset-200, you end up with a network that has 70 million learnable parameters.**

6. Now execute the following script from the Examples directory that does **NOT** use one-hot vectors for representing the words. Instead, this script uses the word2vec word embeddings.

   ```
   text_classification_with_TEXTnet_word2vec.py
   ```

   The most important thing to note here is that now you only have around one million learnable parameters for the same dataset as used earlier. If you run the script for 5 epochs with a learnable rate of 10^{-5}, you are likely to see the following confusion matrix:

   ```
   Number of positive reviews tested: 200

   Number of negative reviews tested: 195

   Confusion Matrix:

                          predicted negative    predicted positive
   true negative:               15.385%               84.615%
   true positive:                 17.5%                 82.5%
   ```

   which is not as dismal as before (although nothing to write home about). If you run the same script on the much larger dataset-200, you get the following results:

   ```
   Number of positive reviews tested: 980

   Number of negative reviews tested: 897

   Confusion matrix:

                          predicted negative    predicted positive
   true negative:               43.81%               56.18%
   true positive:               34.59%               65.40%
   ```

As you would expect, the above result indicates that you improve discrimination with more training data — provided you use fixed-sized embeddings for the words. You could try running the same script with the still larger dataset-400 to see if you can improve upon the result shown above.

With that, you are ready to start working on the homework described in what follows.

3 Tasks

Task 1: As mentioned above, the performance of the `TEXTnet`-based learning framework is not good enough for anything. And, with regard to the performance of the second script, based on `GRUnet`, it is difficult to evaluate it on the same dataset because of the excessively large size of the model. As already stated, the main reason for the large size of the GRU based model is the very large size of the one-hot vectors.

So your first task is to see if there exist any alternatives to the modeling text with one-hot vectors.

Using embeddings is one way to go for efficient numeric representations for the words. [We will discuss word embeddings in the Week 14 lecture.]

While you should feel free to experiment with embeddings, you could also try other more intuitive ways to represent the words numerically. For example, when the vocabulary size is under 2^{16}, why not assign a 16-bit code word to each word in the vocabulary? Give it a try and see what happens.

For another possibility, you could try to represent each word by an integer which would be the index of the word in a sorted list of the vocabulary. If you use this alternative to one-hot vectors, your data for the Amazon reviews would be very much like the time-series data used by Gabriel Loye in his demonstration code for his GRU example in his blog:

`https://blog.floydhub.com/gru-with-pytorch/`

In fact, with this alternative approach to the representation of words by numbers, you could directly use Gabriel Loye's GitHub code to do the sentiment analysis of the 200-dataset. Here is a link to the GitHub code:

`https://github.com/gabrielloye/GRU_Prediction`

Give it a try and see what you get.

Task 2: Could there possibly be other ways to make the size of the model more manageable? To that end, you could try to use the `TEXTnetOrder2` class as a starting point. The performance of `TEXTnetOrder2` is not as bad as that of `TEXTnet`, but only marginally so. How about including additional gating action in `TEXTnetOrder2` to further improve its performance? The `TEXTnetOrder2` class is described on Slides 38 through 43 of the Week13 slides.

Task 3: You will notice that I have used a batch-size of only 1 in the text-related demonstration code in DLStudio. The main reason for that is that when the input to a neural network consists of variable-length sequences, there is no way to batch multiple inputs together. If batching was necessary for better learning, you will have to resort to ploys such as: (1) truncating all of the reviews at, say, the mean length of the reviews; or, (2) padding each review with, say, 0's, so that they call come up to the length of the longest review; or (3) a combination of the first two approaches. Try one or more of these approaches and find out how batching affects the quality of your results.

4 Submission Instructions

- Make sure to submit your code in Python 3.x and not Python 2.x.
- Name the .zip archive as `hw07_<Firstname><Lastname>.zip` (without any white spaces) with your python source files and a summary report in PDF format. The report should have a summary of steps your have tried and your results. **Your code must be your own work.**
- You can resubmit a homework assignment as many times as you want up to the deadline. Each submission will overwrite any previous submission.