BME69500DL ECE695DL
Spring 2020

Homework 6

Deadline : Monday, May 4, 2020, 4:00 pm

1 Introduction

The main goal of this homework is for you 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.

In our case, we will be using such networks for the modeling of variable-length product reviews provided by Amazon for automatic classification of such reviews.

As with the previous homework, this homework also requires you to think outside the box.

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 Version 1.1.2 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.

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.

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

   ```
   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.

5. 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:

   ```
   python text_classification_no_gru.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_200.tar.gz          (vocab_size = 43,285)
   sentiment_dataset_test_200.tar.gz
   ```

   Your results should be similar to those shown on Slide 29 of the Week13 slides. These results are rather poor — on account of the vanishing gradients problem that bedevils neural networks with feedback. In what follows, I'll refer to this dataset as the "dataset-200".

6. Next, execute the following script that uses the GRU based RNN named `GRUnet` for modeling the reviews:

   ```
   python text_classification_with_gru.py
   ```

   You will notice that, as supplied, it will load in the smaller dataset

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

   I will refer to this dataset as the "dataset-40".

   I was not able to use the larger dataset for this script. The larger dataset-200 has a vocabulary size of 43,285, which would also be the size of the one-hot vectors for the words in that dataset. Using GRU, that results in a network with around 70 million learnable parameters. For me personally, it was taking much too long to train such a large network.

   When you execute the above mentioned script, your training loss values should look like what you see in the red plot on Slide 52 of the Week13 slides. Although the rapid decrease in the training loss indicates that good learning is taking place, it does not do us much good because the size of the dataset-40 is much too small.

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

2 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 large size of the vocabulary for the 200-dataset. A vocabulary of size 43,285 results in one-hot vectors for the words being of the same size.

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

Since neural networks can only process numbers,[1] what you need to think of is how to convert text into numbers in such a way that the overall representation of a collection of words does not become excessively large.

For an alternative approach to representing text with numbers, 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? Do you think it might be possible to make the `TEXTnetOrder2` class a bit more elaborate by including some gating action in it so that it performs better than the vanilla `TEXTnet` class? The `TEXTnetOrder2` class is described on Slides 31 through 33 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.

3 Output Format

For each of the three tasks in the homework, pick the best implementation in terms of classification accuracy you are able to achieve and report its result. The output format for all of the three tasks will be the same: printing the training loss every a hundred iterations, followed by the confusion matrix. Redirect the output of your program to a file named "output.txt". Finally, use the following format to structure your output:

```
+Task1:
[epoch:1   iter: 100]      loss: <loss value 1>
[epoch:1   iter: 200]      loss: <loss value 2>
          .
          .
          .
[epoch:1   iter: 100*n]      loss: <loss value n>
[epoch:2   iter: 100]      loss: <loss value 1>
```

[1] On the other hand, traditional programming allows for reasoning framework to be created for purely symbolic data like text.

```
[epoch:2  iter: 200]     loss: <loss value 2>
        .
        .
        .
[epoch:2  iter: 100*n]     loss: <loss value n>
        .
        .
        .
[epoch:N  iter: 100]     loss: <loss value 1>
[epoch:N  iter: 200]     loss: <loss value 2>
        .
        .
        .
[epoch:N  iter: 100*n]     loss: <loss value n>
<confusion matrix result>

+Task2:
[epoch:1  iter: 100]     loss: <loss value 1>
[epoch:1  iter: 200]     loss: <loss value 2>
        .
        .
        .
[epoch:1  iter: 100*n]     loss: <loss value n>
[epoch:2  iter: 100]     loss: <loss value 1>
[epoch:2  iter: 200]     loss: <loss value 2>
        .
        .
        .
[epoch:2  iter: 100*n]     loss: <loss value n>
        .
        .
        .
[epoch:N  iter: 100]     loss: <loss value 1>
[epoch:N  iter: 200]     loss: <loss value 2>
        .
        .
        .
[epoch:N  iter: 100*n]     loss: <loss value n>
<confusion matrix result>

+Task3:
[epoch:1  iter: 100]     loss: <loss value 1>
[epoch:1  iter: 200]     loss: <loss value 2>
        .
        .
        .
[epoch:1  iter: 100*n]     loss: <loss value n>
[epoch:2  iter: 100]     loss: <loss value 1>
[epoch:2  iter: 200]     loss: <loss value 2>
        .
        .
        .
[epoch:2  iter: 100*n]     loss: <loss value n>
        .
        .
        .
[epoch:N  iter: 100]     loss: <loss value 1>
```

```
[epoch:N  iter: 200]      loss: <loss value 2>
        .
        .
        .
[epoch:N  iter: 100*n]      loss: <loss value n>
<confusion matrix result>
```

4 Submission

- Make sure to submit your code in Python 3.x and not Python 2.x.
- Name your main Python file as hw06.py
- All homework to be submitted on-line through the *turnin* command on the min.ecn.purdue.edu[2][3] server.
- Log into min.ecn.purdue.edu server using your Purdue career account.
- From min.ecn.purdue.edu, go to the directory where your homework files are. e.g., if your files are at /home/hw1 directory then go to /home/hw1. Obviously, if you don't have your files on the server, move them there using, for example, the *scp* command.
- Type in the following command:

  ```
  turnin -c ecedl -p hw<double digits hw#> <your files separated by a space>
  ```

- As an example, for this homework you will enter the command:

  ```
  turnin -c ecedl -p hw06 hw06.py output.txt
  ```

- You should get a statement that says "Your files have been submitted to ecedl, hw06 for grading". You can verify your submission by typing:

  ```
  turnin -c ecedl -p hw<double digits homework number> -v .
  ```

[2] If you are an undergraduate student, use the shay.ecn.purdue.edu server instead of the **min** server
[3] If you registered this class as the BME695 course then use the weldon.ecn.purdue.edu server