**CS6375 Assignment 1**

<https://github.com/seanokeefe419/CS6375Assignment1>

Sean O’Keefe

sto210000

**1. Introduction and Data**

This assignment is about using a Feedforward Neural Network (FFN) and a Recurrent Neural Network (RNN) for sentiment analysis. The dataset used for this assignment consisted of reviews from Yelp. Each review had text and a rating from 1 to 5, inclusive, associated with it, and the reviews were divided into a training set, a validation set, and a testing set. The goal for the neural networks was to predict the rating for a review given its text. The effectiveness of the neural networks would be evaluated using accuracy scores, which would be calculated by dividing the number of correct review ratings by the total number of review ratings.

The below table shows the three sets of reviews and the number of reviews in each set.

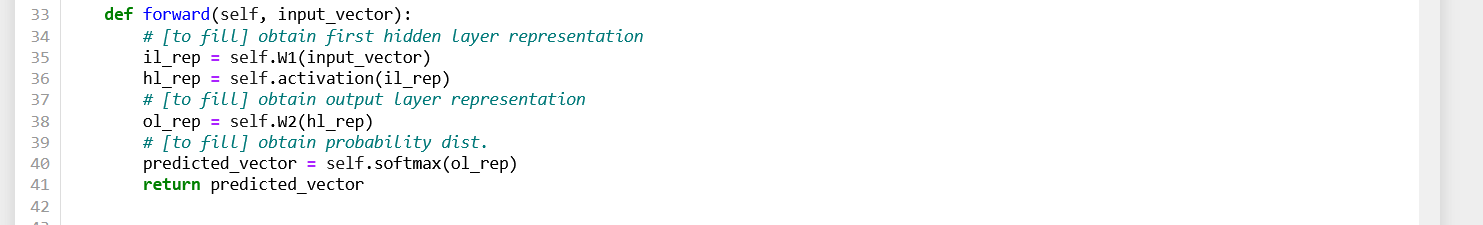
|  |  |
| --- | --- |
| **Set** | **Number of Reviews** |
| Training | 16000 |
| Validation | 800 |
| Testing | 800 |

**2. Implementations**

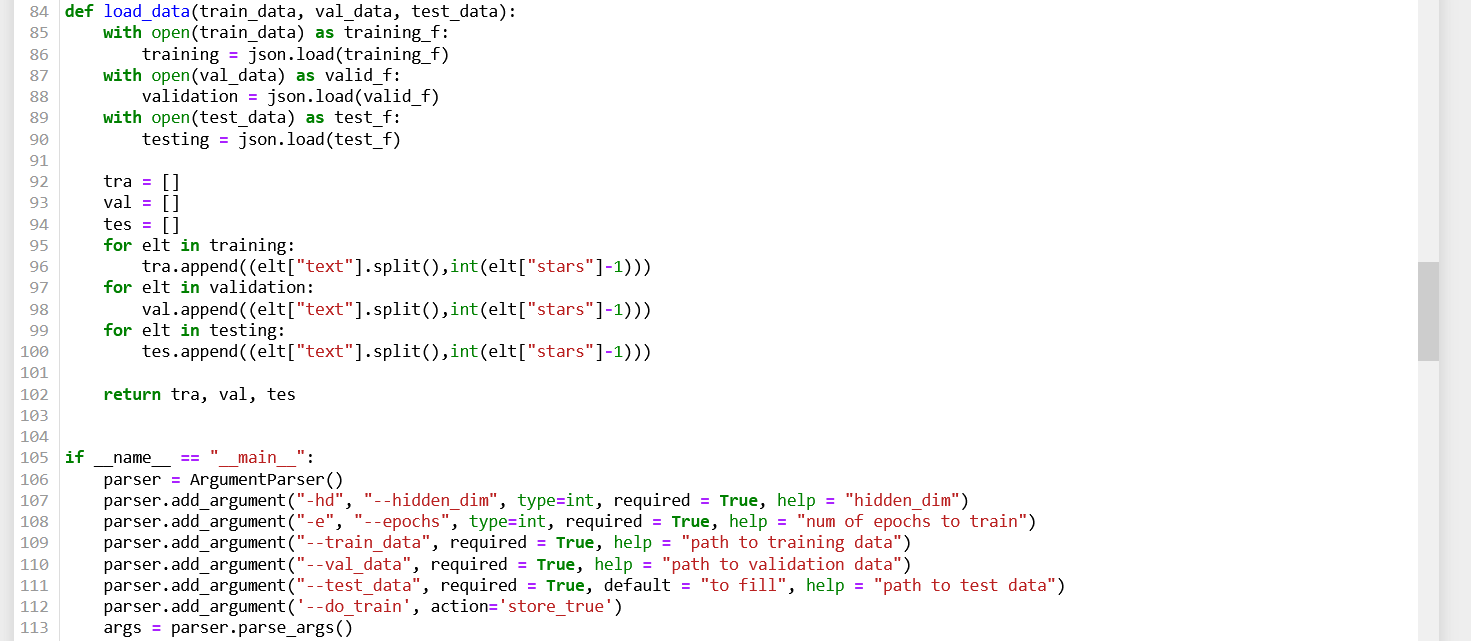
**2.1 FFNN**

For the Feedforward Neural Network, I was given an incomplete Python code file, ffnn.py, and had to complete it. Mostly, this was just filling out the forward function in the FFNN class in the file. I used self.W1 under \_\_init\_\_ with the forward function’s input\_vector argument to represent the initial layer, and then used self.activation with this initial layer representation to represent the first hidden layer. After that, I used self.W2 with the first hidden layer representation to represent the output layer. Finally, I used self.softmax, which actually uses LogSoftmax instead of Softmax, with the output layer representation for obtaining the probability distribution, and then returned the vector that the LogSoftmax produced.

In the below screenshot, the finished forward function can be seen beginning on line 33 and ending on line 41.

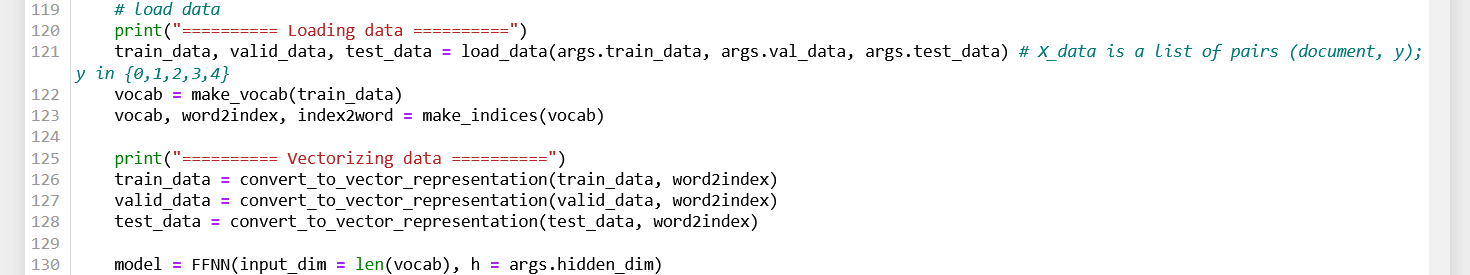


I had to make a few other changes to the file. These had to do with adding in functionality for the testing set. The original file only had code for the training set and the validation set. As seen in the below screenshot, I had to change the load\_data function. I added a third argument, test\_data, and a third return, tes, and also added lines that did the same things with the test data as for the other two data sets. These changes are shown in the below screenshot.



Under “if\_\_name\_\_ == “\_\_main\_\_””:, I had to make test\_data a required argument, as seen on line 111 in the above screenshot.

I had to make more changes to the code to make it do the same things with the test data as with the evaluation data, which can be seen on lines 121, and 128 in the first screenshot, and lines 192-216 in the second screenshot.



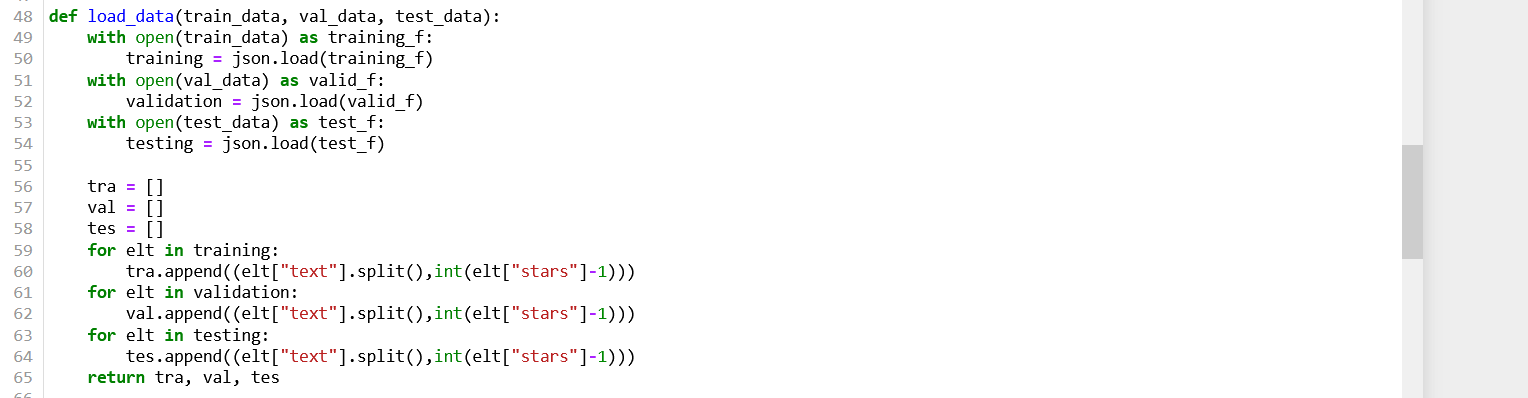


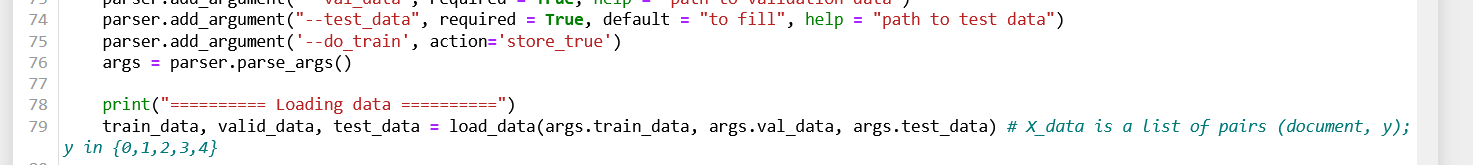
**2.2 RNN**

For the RNN, I similarly had to complete the forward function. I used torch.randn to represent the hidden layer, and used self.rnn with the inputs argument and the hidden layer representation followed by self.W with the output of the preceding to represent the output layer. I had trouble figuring out what was meant by “sum over output”, so the code in that section is commented out. Finally, I used self.softmax (which again actually used LogSoftmax) on the output layer representation to get a vector of predictions that would then be returned. The screenshot below shows this new forward function.

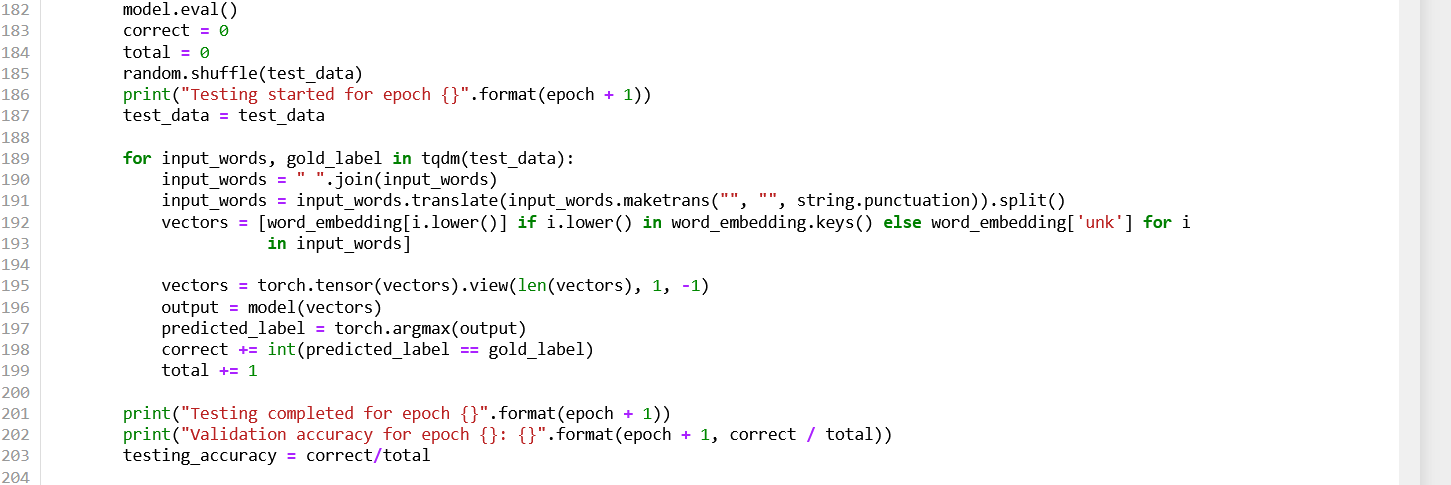


Similarly to the FFNN, I had to add in functionality for the test data set in various places in the code. The below code snippets show these additions.









**3. Experiments and Results**

**Evaluations**

To evaluate these models, I used accuracy scores for the training set data, the validation set data, and the testing set data. These accuracies were calculated by dividing the total number of correct review ratings predicted by the models for a given set by the total number of ratings in that same set.

**Results**

The FFNN had two hyperparameters: number of hidden dimensions and number of epochs. I ran the model with various combinations of these hyperparameters and compared the results. The results are shown below in the table. Because random.seed and torch.manual\_seed are always set to 42, these results will be consistent.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Trial | Hidden Dimensions | Epochs | Training accuracy | Validation accuracy | Testing accuracy |
| 1 | 8 | 5 | 0.5825 | 0.4975 | 0.5625 |
| 2 | 16 | 5 | 0.5856875 | 0.53875 | 0.455 |

As can be seen, increasing the hidden dimensions from 8 to 16 while keeping the epochs constant had little effect on the training accuracy, but increased the validation accuracy slightly and significantly dropped the testing accuracy. It seems as though having less hidden dimensions makes the model adapt better to new testing data.

I was unable to get the RNN working correctly. I think this had to do with my struggles with the “sum over output” section. If I ran the rnn.py file, it would get stuck in an infinite while loop because the training accuracy and evaluation accuracy would never change, preventing stopping\_condition from ever being True. Also, every time I ran the code, the values for the three accuracies would always be the same even though there was no seed set.

Since RNN never worked properly, I cannot provide meaningful analysis for its results.

**4. Conclusion**

I worked on this assignment by myself. I thought the assignment was difficult and took a long time to complete, particularly for the RNN because I could not get it working properly despite multiple attempts.