Ryan Morrell and Danny Kim

NLP Programming Assignment 2

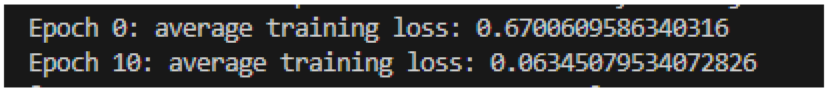
Code available at: https://github.com/PlutoPotatoes/Sentiment-Analysis-Project

Implementation:

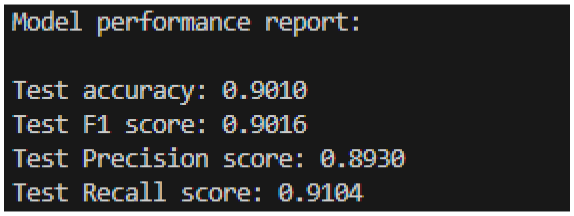
The Neural Network architecture is as follows: an embedding layer, a hidden layer, an activation layer, and an output layer.

- The embedding layer's input shape is [batch, seq_len] and outputs [batch, seq_len, embedding_dim]. The goal of this layer is to map tokens from the vocab to vectors.
- A two sets of linear-activation layer pairs made up of:
    - A hidden layer performs a linear transformation on each embedded token vector to output shape [batch_size, sequence_len, hidden_size].
    - Then a ReLU activation layer introduces non-lineararity into the network to mitigate the vanishing gradient problem.
- Then the hidden representations are combined into one vector (in forward(self, x)), producing an output size of [batch, hidden_size].
- Finally, the output layer maps the hidden layer's output to the final output dimension [batch, output_size] where output_size is equal to the number of possible classes.

Performance:



```
Epoch 0: average training loss: 0.6700609586340316
Epoch 10: average training loss: 0.06345079534072826
```



```
Model performance report:

Test accuracy: 0.9010
Test F1 score: 0.9016
Test Precision score: 0.8930
Test Recall score: 0.9104
```

- Optimizer
    - optim.Adam optimizer was used during the training of this model. This optimizer was used because it generally converges faster then standard SGD.
- Loss Function

    - nn.CrossEntropyLoss was used as the loss function. This loss function was used because it reflects the standard loss function for classification models.

- Hyperparameters:
    - Learning rate: 0.001
    - Batch size: 64
    - Epochs: 20 for testing 30 in final implementation

Implementation Commentary:

Our implementation started with a single embed -> linear -> activation -> linear output structure both as a proof of concept and to test the capabilities of the model at its simplest. This basic implementation worked to an extent and was able to reach high f1 scores during training after converging extremely quickly. However, the basic model struggled to correctly classify the unlabeled test data. We tried a few solutions to fix this starting with shuffling the training/validation data (See the problems section) and tuning hyperparameters before finally deciding to add another linear and activation layer to the network. With this second layer the model's f1 scores dropped slightly but it was able to correctly classify far more of the unlabeled data. The second layer seemed especially helpful for classifying the longer reviews in the dataset as those contained more contradictory phrasing due to their depth.

Problems During Development:

One problem we had was that when utilizing the training data for training and then testing it on the testing data, the model was very overfitted, thus resulting in all performance reports being 1. This led us to believe that the model wasn't actually learning but rather it was memorizing. We were curious if the structure of the training data, which listed all negative reviews first and then all positive reviews, was having an impact on the model. To address this issue, we first combined the training and testing dataset, shuffled it randomly, then created a new training/testing dataset with an 80/20 split. This ensured the model wasn't just memorizing the training set which probably contained overlapping or similar data. After shuffling our training history more closely reflected the gradual improvements we expected to see from the model. This did have a slight impact on the predicted results and the model was able to correctly classify a few more reviews; however, we still needed to expand the network to see large improvements in classification accuracy.

Conclusion:

This project demonstrates that a relatively simple neural network can achieve high performance on a sentiment analysis task. By identifying and correcting the data mix-up and expanding our simple model, we were able to train a model that produced valid results at 90.4% accuracy in training with objectively similar results in unlabeled testing.