Ονοματεπώνυμο: Παντελεήμων Μαλέκας
A.M: 1115201600268

1. For the first question a bidirectional stacked RNN model was defined. The preparation and preprocessing of the data is for the most part the same as in the GloVe and One-Hot models of HW2. The main difference here was that I was able to utilize the entirety of the dataset and the vocabulary didn't need to be reduced. In HW2 using the entire dataset and all the tokens of vocabulary slowed way down the execution and gave problems regarding the RAM of Google Collab (also, I wasn't aware of the existence of CUDA…). The vocabulary is created using a GloVe Twitter embedding and is then inserted through a nn.Embedding layer.

Important note: This assignment makes use of the CUDA functionality. Make sure to activate the GPU before executing any cells of the notebook.

The final model had the given parameters: Batch size = 2500. Embedding size = 300. Learning rate = 0.002. Cell type = GRU. Hidden dimension = 128. Hidden Layers = 4. Skip connections = No. Gradient Clipping = Yes. Dropout Probability = 0.9

A few comments about each parameter:

The batch size is 2500. A smaller size would decrease the speed of the model because it required more iterations. A higher size also decreased the speed of the model because it required larger array operations in each iteration.

Embedding size is 300 because that is a good number for the vocabulary we have.

The learning rate is 0.002. Any number higher or lower than that was subject to overfitting.

The cell type is GRU. Although the LSTM is supposed to be a more effective model than GRU this didn't seem to be the case here. With all the other parameters being the same, the GRU model always gave better scores in a faster time. Example: For the given parameters the GRU model finishes in 418.28 seconds and gives an average score of 87% (more on that later). The LSTM model on the other hand finishes in 473.338 seconds with an average score of 84%. Although the LSTM model is expected to be a bit slower, the scores weren't getting any better. Maybe with a few different parameters it could achieve a bit higher accuracy, but the GRU seemed to be the best for these ones.

The hidden dimension is 128. Any number lower than that improved the accuracy of the model very slightly. Any higher number slowed down the speed and didn't improve the scores that much either.

The hidden layers are 4. The reasons this number was chosen are pretty much the same ones as for the one in the hidden dimensions.

Skip connections functionality was not used in this model. I tried implementing this feature by adding the embedding output in the output of the linear layer (the addition is commented out in case one wants to re-enable it). I decided not to use it in this model since it didn't appear to increase the model's accuracy (if anything there was a slight decrease).

Gradient clipping is used in this model. Although there was only a very slight increase in each epoch it appeared to be beneficial.

Dropout probability is 0.9. Numbers lower than that didn't increase the model's accuracy that much. Also, I decided to add the dropout probability between the RNN and linear layers. I also tried adding a dropout probability in between the hidden layers but this didn't appear to impact the model's accuracy in any way.

Now we'll take a look at the results of each epoch and the average Precision, Recall and F1 scores.

Epoch	Training loss	Test loss	Test accuracy
1	0.527	0.433	0.799
2	0.482	0.405	0.818
3	0.448	0.371	0.837
4	0.415	0.338	0.856
5	0.383	0.307	0.871

We can see that the training and test losses actually decrease which means that the model appears to improve with each epoch. This can be also seen visualised in the Loss vs Epochs plot in the notebook. Also, the test accuracy increases over time. We can see that the accuracy starts off quite high too. By the final epoch we have quite an improvement.

Precision	87.15%
Recall	87.12%
F1-Measure	87.12%

As we can see the given scores are pretty much the same as the accuracy score of the last epoch. A classification report on specifically the last epoch predictions can be seen in the notebook.

A few comments about the Loss vs Epochs and ROC plots:

In the ROC curve, we can see the expected curve for an AUC score of ~93%, which was also given from the scores in the classification report.

In the Loss vs Epochs, we can see that the loss starts off quite high, but as the epochs progress the loss decreases. Which shows that the model actually improves with each epoch. I chose to plot two variants. The left one shows the curves of the average losses of the train and test sets. Since the curves appeared to be 'too smooth', I decided to plot the losses on each batch of every iteration to show how each epoch affects the datasets.

Finally, here's a comparison with the Feed Forward GloVe model from Homework 2. I should note that I chose the GloVe model for this comparison, even though I chose a different final model in the second homework. This was done because the final model in Homework 2 was a TF-IDF model that didn't make use of the torchtext library, the splitting into batches etc. The GloVe model on the other hand was developed similarly with the one for this assignment, so it was chosen for the comparison.

	Bidirectional stacked GRU	Feed Forward GloVe
Precision	87.15%	77.58%
Recall	87.12%	73.16%
F1-Measure	87.12%	73.83%

As we can clearly see the RNN model is definitely a huge improvement compared to the FF GloVe model. This is expected of course, given that the Feed Forward model is simpler. I should note that the dataset in Homework 2 could not be used in its entirety, therefore the specific numbers are not the best ones for the comparison. However, even with the smaller dataset used in the previous assignment, the RNN model still achieved higher accuracy scores, which makes the comparison consistent enough.

Links used for the first question:
https://github.com/bentrevett/pytorch-sentiment-analysis/blob/master/2%20-%20Upgraded%20Sentiment%20Analysis.ipynb

https://medium.com/@lamiae.hana/a-step-by-step-guide-on-sentiment-analysis-with-rnn-and-lstm-3a293817e314

https://towardsdatascience.com/text-classification-with-pytorch-7111dae111a6

https://towardsdatascience.com/lstm-text-classification-using-pytorch-2c6c657f8fc0

https://discuss.pytorch.org/t/resolved-gru-for-sentiment-classification/1660