50.040 Design Project Report

By: Lim Zhen Dong, Arnold (1004117); Koh Jun Hao (1004295)

Part 2

Results on dev.in:

```
processed 3809 tokens with 210 phrases; found: 168 phrases; correct: 66.
accuracy:  28.62%; (non-O)
accuracy:  92.57%; precision:  39.29%; recall:  31.43%; FB1:  34.92
         negative: precision:  32.26%; recall:  15.38%; FB1:  20.83  31
          neutral: precision:   7.14%; recall:  12.50%; FB1:   9.09  14
         positive: precision:  44.72%; recall:  40.15%; FB1:  42.31  123
(39.285714285714285, 31.428571428571427, 34.920634920634924)
```

Part 4

Immediate Loss: 3546.39
Final Loss (after training): 1817.95
Number of iterations: 195
Training time: ~26 minutes

Results on dev.in:

```
processed 3809 tokens with 210 phrases; found: 229 phrases; correct: 79.
accuracy:  35.02%; (non-O)
accuracy:  90.39%; precision:  34.50%; recall:  37.62%; FB1:  35.99
         negative: precision:  29.27%; recall:  18.46%; FB1:  22.64  41
          neutral: precision:   0.00%; recall:   0.00%; FB1:   0.00  27
         positive: precision:  41.61%; recall:  48.91%; FB1:  44.97  161
(34.49781659388647, 37.61904761904762, 35.99088838268793)
```

Part 5

Results on dev.in:

```
processed 3809 tokens with 210 phrases; found: 218 phrases; correct: 100.
accuracy:  41.08%; (non-O)
accuracy:  92.94%; precision:  45.87%; recall:  47.62%; FB1:  46.73
         negative: precision:  35.71%; recall:  23.08%; FB1:  28.04  42
          neutral: precision:  16.67%; recall:  37.50%; FB1:  23.08  18
         positive: precision:  51.90%; recall:  59.85%; FB1:  55.59  158
(45.87155963302752, 47.61904761904761, 46.728971962616825)
```

<u>Part 6 - Design Challenge</u>

(i) On top of the previous 5 features used in Part 5, we used the following new features:

- Bigram: $y_{i-1} + y_i + x_{i-1}$
- Bigram: $y_{i-1} + y_i + x_{i+1}$

We also attempted to use the below feature:

- Token Transitions: $y_{i-1} + x_{i-1} + x_i$

However, we found that token transitions negatively impacts the performance of the model and as such had removed it

Results on dev.in:

```
processed 3809 tokens with 210 phrases; found: 204 phrases; correct: 94.
accuracy:  38.72%; (non-O)
accuracy:  93.04%; precision:  46.08%; recall:  44.76%; FB1:  45.41
         negative: precision:  32.69%; recall:  26.15%; FB1:  29.06  52
          neutral: precision:  15.38%; recall:  25.00%; FB1:  19.05  13
         positive: precision:  53.96%; recall:  54.74%; FB1:  54.35  139
(46.07843137254902, 44.761904761904766, 45.41062801932367
```

(ii) We implemented a Long Short Term Memory (LSTM) model using PyTorch external package. The LSTM model is a deep learning model which implements both feedforward and feedback connections, allowing it to work with sequential data. Each unit comprises a cell, which takes in both new data point at current time step and output data from the previous time step, and outputs a new cell state along with the output data. It is an extension of recurrent neural networks, covering the vanishing/exploding gradient and short term memory weakness of RNNs by maintaining a long running cell state which allows it to remember long term dependencies.

To apply it on aspect-based sentiment analysis, the LSTM model takes in a sequence of tokens $x_1, x_2, x_3,$ .... and outputs a sequence of tags $y_1, y_2, y_3,$ ...., which represents the different sentiments.

To work with the model, it is required that the tokens and tags are converted into indices. We used Torchtext.Vocab to create the vocabulary for both tokens and tags, while also including other special tags - namely START, STOP and PAD. Furthermore, the model requires a fixed sequence length to train - as such, the data was further preprocessed by adding paddings to the sequences to ensure a fixed length for all the sequences in each batch. The fixed length is set at 75, given that the maximum sequence (sentence) length we found was 74. Finally, the data was loaded into two Torch.Dataloader objects with a batch size of 32 - one for training and other for validation.

Our model architecture is as follows

- **Embedding layer**: to embed the input sequence (integers) into an embedding of fixed size
- **Bidirectional LSTM**: Long Short Term memory unit. We set the number of units to be 3
- **Linear layer**: Fully connected layer which allows us to map the output of the LSTM onto a set output size
- **Sigmoid**: Sigmoid activation function which produces probabilities for different tags for each single token

We used bidirectional LSTMs to allow the model to learn from both in front and behind a specific token, as it allows the input to flow in forward and backward direction. This provides better context for the model, which could improve its performance in predicting the tags. Furthermore, we implemented dropout (one of the hyperparameters in Pytorch's LSTM API) at 0.3, which means that for each training iteration, 0.3 of the parameters in the LSTM would be fixed and not affected by the backpropagation of gradients. By doing so, it reduces the chance of overfitting, allowing the model to generalize better to other data.
To train the model, Adam optimizer, which is an adaptive gradient algorithm that also adapts well to individual parameter learning rates, is used. The loss function is Cross Entropy Loss, given that the output of the model is a set of probabilities for each class (tag). In total there are 10 tags to predict - asides from the seven different sentiments (O, B-positive, B-negative, B-neutral, I-positive, I-negative, I-neutral), there is 'START', 'END' and 'PAD', these three of which the token and tag are the same fixed mappings.

At each training step, the model is trained on the train dataloader, before the validation dataloader is passed through it to get the validation loss. The current epoch is printed to the console, along with the training and validation loss. There is early stopping implemented, which checks for whether the validation loss has not improved for a set number of epochs (we set it at 10) - and if so, stop the training and set the model to be the model with the lowest validation loss. Through early stopping, we ensure that the output model will always be the best possible model and the chance of overfitting is greatly reduced as the model is not continuously training and overfitting on the training set.

The performance of our LSTM model is as follows:

- **Precision:** 43.26%
- **Recall:** 36.67%
- **F1 score:** 39.69%

Results on dev.in:

```
processed 3809 tokens with 210 phrases; found: 178 phrases; correct: 77.
accuracy:  29.97%; (non-O)
accuracy:  93.31%; precision:  43.26%; recall:  36.67%; FB1:  39.69
         negative: precision:   0.00%; recall:   0.00%; FB1:   0.00  0
          neutral: precision:   0.00%; recall:   0.00%; FB1:   0.00  0
         positive: precision:  43.26%; recall:  56.20%; FB1:  48.89  178
(43.258426966292134, 36.666666666666664, 39.69072164948454)
```