ECE324: Assignment 5

Subjective/Objective Sentence Classification

Spencer Ball #1004762599

3 Preparing the data

3.1 Create train/validation/test splits

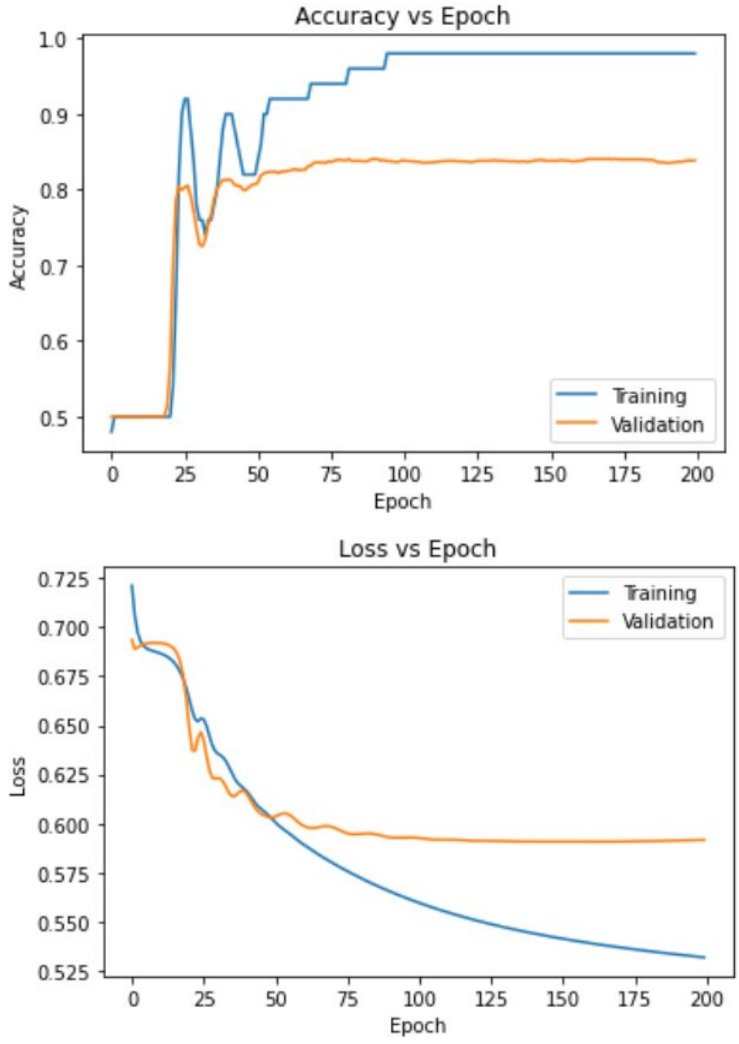
Here are the shapes of the balanced, separated datasets:

```
training set, objective: (3200, 2)
training set, subjective: (3200, 2)
validation set, objective: (800, 2)
validation set, subjective: (800, 2)
test set, objective: (1000, 2)
test set, subjective: (1000, 2)
overfit set, objective: (25, 2)
overfit set, subjective: (25, 2)
```

4 Baseline Model and Training

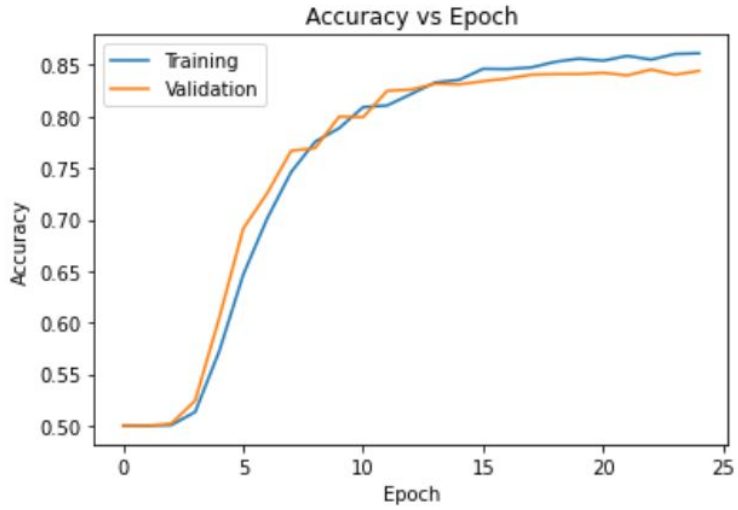
4.4 Overfitting to debug

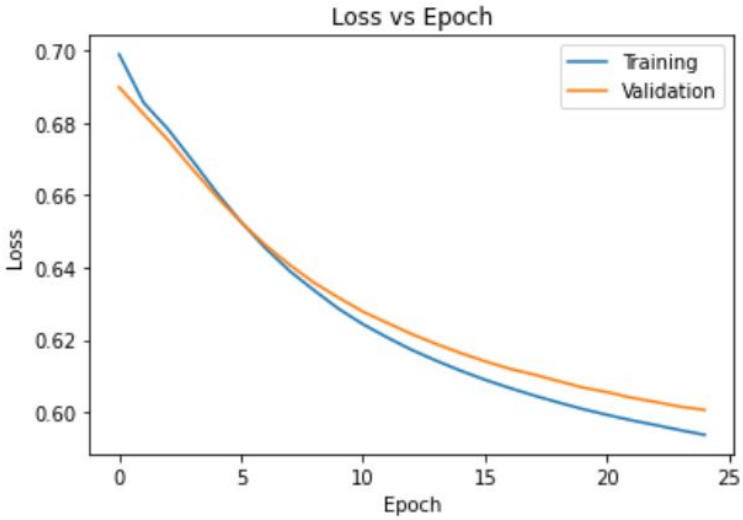
Loss and accuracy plots from training on 50 examples (overfit.tsv) and validating on entire validation set (valid_data.tsv):
(lr = 0.05, epochs = 200, batch size = 64)



4.5 Full Training Data

Loss and accuracy plots:
Using hyperparameters from Table 1



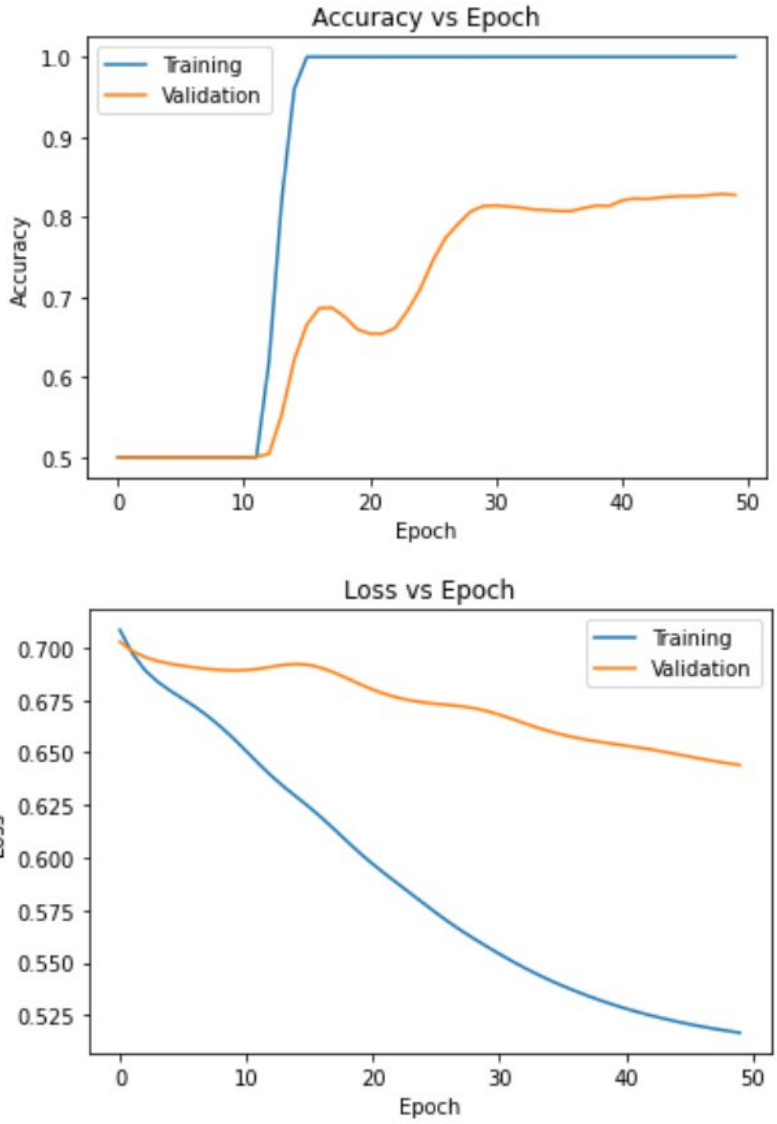



```
Highest Validation Accuracy: 0.845625
Average Test Accuracy: 0.88720703125
```

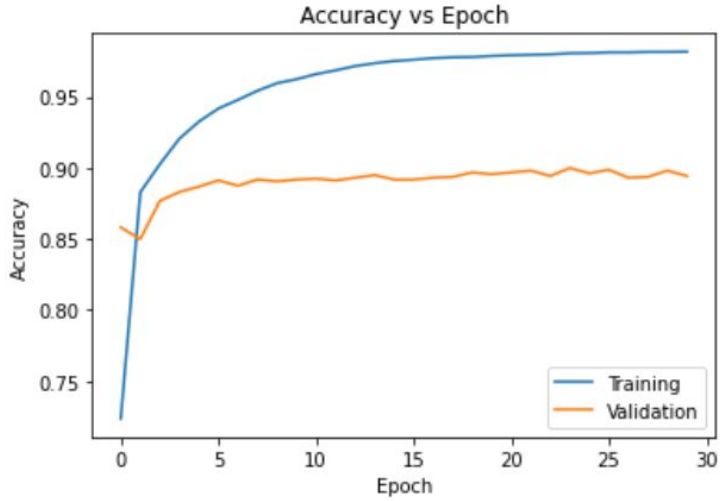
5 Convolutional Neural Network (CNN)

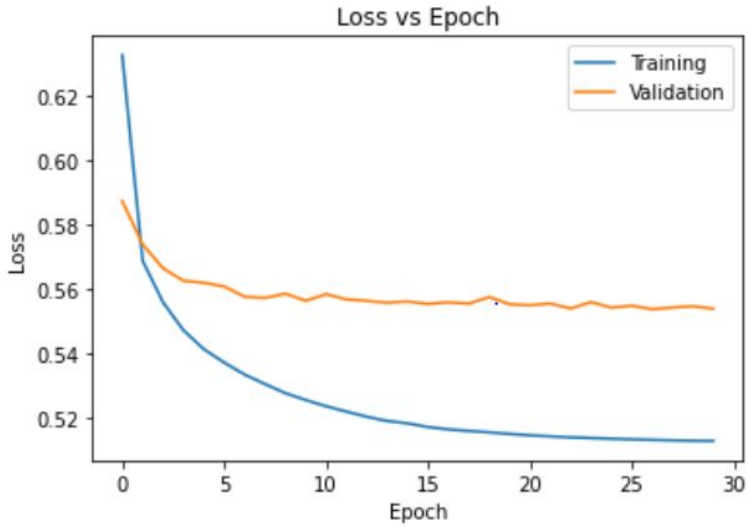
5.1 Overfit, Training, and Test

Overfitting: Loss and accuracy plots from training on 50 examples (overfit.tsv) and validating on the entire validation set (valid_data.tsv).
(lr = 0.001, epochs = 50, batch size = 64)



Training & Testing: Loss and accuracy plots from training on full training dataset.
*(lr = 0.001, epochs = 30, batch size = 64) **Note**: shifted y-axis starting at 0.75*



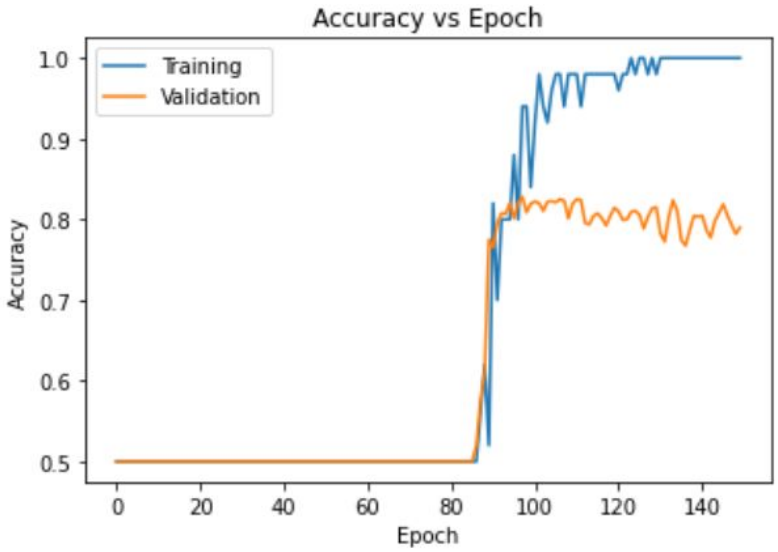


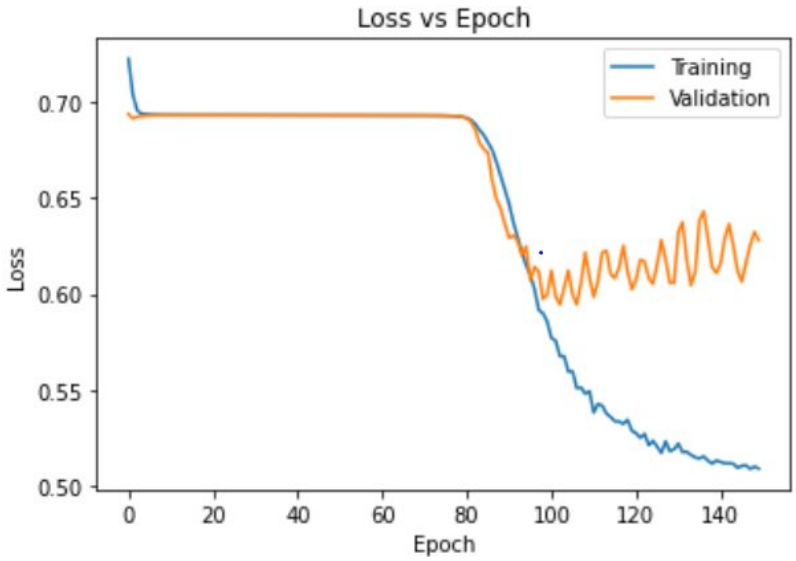

```
Highest Validation Accuracy: 0.9
Average Test Accuracy: 0.923828125
```

6 Recurrent Neural Network (RNN)

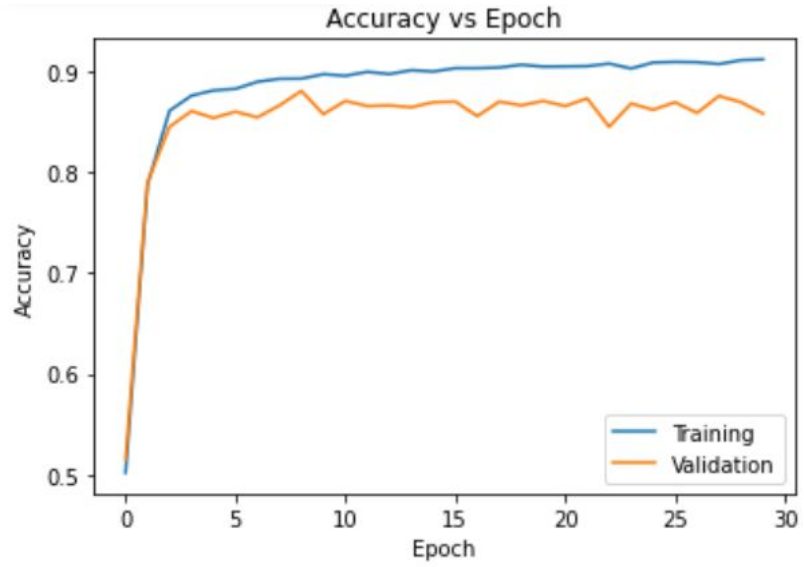
6.1 Overfit, Training, and Test

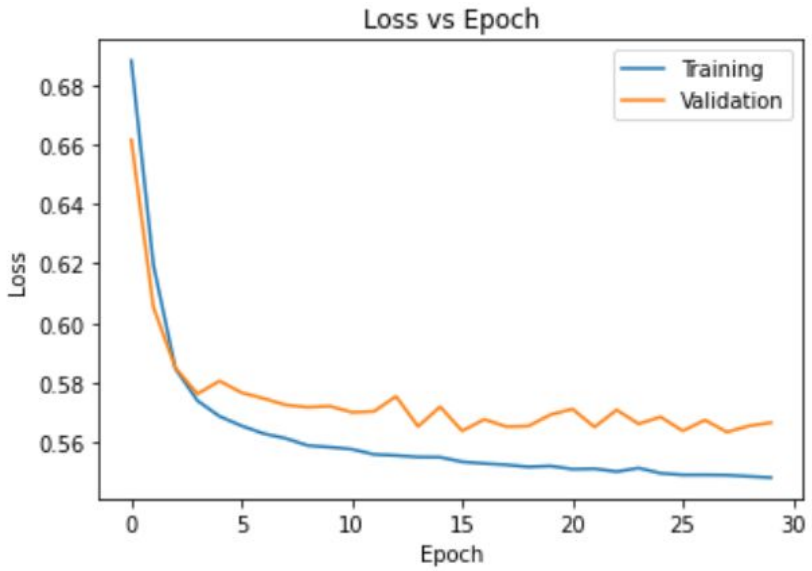
Overfitting: Loss and accuracy plots from training on 50 examples (overfit.tsv) and validating on the entire validation set (valid_data.tsv).
(lr = 0.01, epochs = 150, batch size = 64)





Training & Testing: Loss and accuracy plots from training on full training dataset.
(lr = 0.001, epochs = 30, batch size = 64)






```
Highest Validation Accuracy: 0.880625
Average Test Accuracy: 0.89111328125
```

8 Experimental and Conceptual Questions

Model	Train Acc.	Valid Acc.	Test Acc.	Train Loss	Valid Loss	Test Loss
Baseline	86.19%	84.56%	88.72%	0.5936	0.6003	0.5835
CNN	98.30%	90.00%	92.38%	0.5122	0.5539	0.5443
RNN	91.02%	88.06%	89.11%	0.5485	0.5641	0.5546

1. The CNN model performed the best, reaching the highest accuracies and lowest losses across training, validation, and testing.

Yes, there is a significant difference between the validation and test accuracies for all three models. This difference exists because the training set is more similar to the test set than it is to the validation set causing the model to make better predictions when given examples from the test set versus from the validation set. The discrepancies between the validation set and test set are a product of the random splitting that was done to preprocess the data. Evidently, the random splitting process ended up putting sentences with more similar characteristics in the training and testing sets allowing the model to perform better on the test set.

2. The ordering of words in the input sentence is being ignored in the baseline model. For example, say the user inputs the sentence "*That movie is amazing and has good characters*" into the baseline model and it returns prediction **x**. If the user then input the sentence "*amazing good and is That movie characters has*", the baseline model would return the identical prediction **x** because the words in both sentences are identical and the baseline model operates on an average of all the word vectors across an entire sentence. Therefore lots of meaning which arises from the structure of the sentence is lost when using the baseline model.

When the baseline model outputs a value that is completely off of both the CNN and RNN predictions but those latter two predictions are quite close together it is clear that the ordering of words in the sentence contains some meaning that was missed by the baseline model.

3. (a) Default scenario.
(lr = 0.001, epochs = 30, batch size = 64)

Model	**Train Acc.**	**Valid Acc.**	**Test Acc.**	**Train Loss**	**Valid Loss**	**Test Loss**
RNN	91.02%	88.06%	89.11%	0.5485	0.5641	0.5546

(b) Without calling pack_padded_sequence.
(lr = 0.001, epochs = 30, batch size = 64)

Model	**Train Acc.**	**Valid Acc.**	**Test Acc.**	**Train Loss**	**Valid Loss**	**Test Loss**
RNN	91.02%	87.63%	90.33%	0.5485	0.5641	0.5546

(c) Without calling pack_padded_sequence, using Iterator instead of BucketIterator.
(lr = 0.001, epochs = 30, batch size = 64)

Model	**Train Acc.**	**Valid Acc.**	**Test Acc.**	**Train Loss**	**Valid Loss**	**Test Loss**
RNN	90.53%	88.56%	91.02%	0.5524	0.5555	0.5489

The effect of not calling pack_padded_sequence is not very noticeable while BucketIterator is being used because BucketIterator ensures all sentences in a given batch are the same length (or very close). This effectively removes the need for padding between examples because the final hidden state taken from each sentence is the correct final hidden states as most, if not all the sentences in each batch, are the same length.

When both pack_padded_sequence and BucketIterator are removed, then the training accuracy begins to suffer. This makes sense because the final hidden states from any sentence shorter than the longest sentence in the batch are incorrect/lost.

4. The CNN kernels are learning to detect elements of subjectivity/objectivity in the groups of 2 or 4 words which they scan.

When doing the MaxPool operation, the model is discarding the outputs from the kernels which were less useful for determining the objectivity/subjectivity of the sentence. The smaller the value of the output, the less information it provides about the nature of the sentence and therefore the more likely it is to be discarded by the MaxPool operation.

This is different from the baseline model's way of discarding information because the CNN still retains information regarding word order in the sentence while the word vector average computed by the baseline model does not.

5.

Definitely objective:

```
The sun rises every day.
Model baseline: objective (0.254)
Model CNN: objective (0.006)
Model RNN: objective (0.003)
```

Definitely subjective:

```
That car looks cool.
Model baseline: subjective (0.936)
Model CNN: subjective (1.000)
Model RNN: subjective (0.999)
```

Borderline objective:

```
Today it is hot outside.
Model baseline: subjective (0.675)
Model CNN: subjective (0.928)
Model RNN: objective (0.230)
```

Borderline subjective:

```
That car is well designed.
Model baseline: subjective (0.653)
Model CNN: subjective (0.857)
Model RNN: objective (0.401)
```

The baseline model had the least extreme output values, implying it was the least certain of its predictions. This is as expected as it uses the least amount of information to train itself and to make its predictions meaning it will make the least informed and therefore the least certain classifications.

The CNN and RNN both work very well when classifying definitely objective and definitely subjective sentences. When given a borderline sentence, the CNN makes more extreme choices, predicting quite far in whichever direction it chooses. In contrast, when given a borderline sentence the RNN makes more moderate choices. Neither model is better at classifying these borderline sentences and this should be expected as even a group of people would probably not agree on the classification of many borderline sentences.

Given obvious sentences, the models all agree with each other and they are correct. With less obvious sentences, the models do not all agree with each other and the majority vote is not always correct either.

In general, the RNN model seems to be performing best. I say this because its predictions fall in line with my own intuition more often than the predictions from the other two models do.

6. (a) I spent about 10-12 hours on assignment 5.

(b) I found building the CNN and the RNN models most challenging as I kept getting tensor shape incompatibility errors which were difficult to debug.

(c) I enjoyed playing around with the subjective bot quite a lot.

(d) The instructions for building the CNN and RNN were too vague in my opinion and there was a lot that I had to figure out on my own. Using different suggestions from StackOverflow and various other sources became very confusing at multiple points.

(e) The CNN and RNN diagrams in the project document were helpful and I believe there should be more visual aids like those included in the instructions.