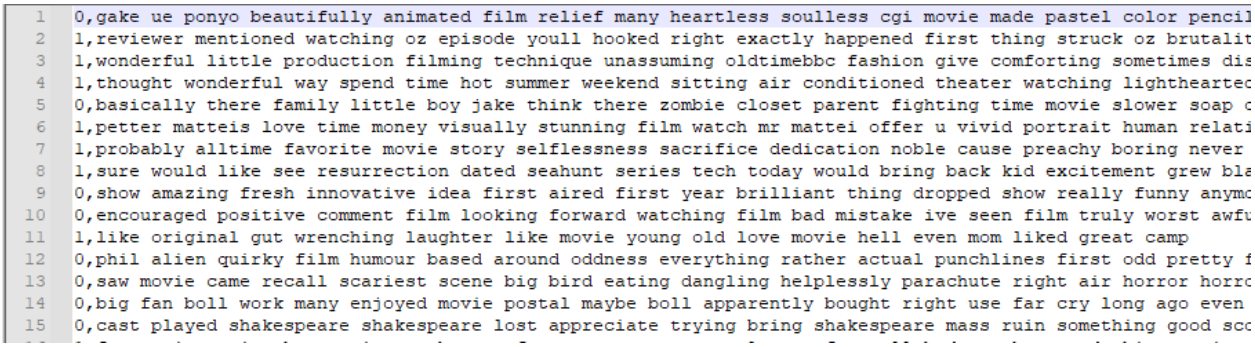
Travis Browning
University of Wolverhampton
Machine Learning, L7N008
Dr. Burcu Can
Assignment 3
17 June 2021

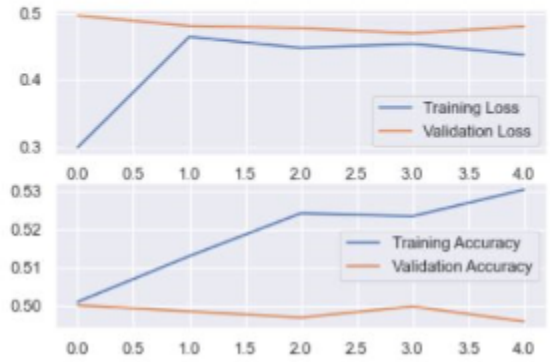
In approaching task 1, the first thing I did was have a look at the data set. At a glance the data set appeared to be mostly readable text. Newlines appeared to be written with HTML break tags. Rather than opening a Python HTML library, I decided to use Regular Expressions to filter the tags.

I used a for loop to work through the 50 thousand entries of the spreadsheet. Mindful of memory use, I opened a writer to write lines to a new spreadsheet as they were processed. For every row in the original spreadsheet there are two columns. The first column contains the entire text of the review, and separated by the comma is a binary value of sentiment score, either positive or negative. The filtering in task 1 was done on the first column of data. Using a count to bypass the first header row, the text of all other rows were filtered by Regular Expression for break tags, digits, and punctuation. Text was lowercase and tokenized. Running the text through the preprocessing at this point took roughly ten minutes, due to the large quantity of entries. The next two added steps of preprocessing, filtering for stopwords and lemmatizing, dramatically increased the processing time. Utilizing NLTK's stopword corpora and Wordnet also through NLTK, articles are read token by token. Tokens contained in Stopwords are omitted, and remaining tokens are lemmatized using wordnet. Running task 1, at this point took a Jupyter notebook 110180 seconds, or just under 31 hours. I attempted running the same task in Colab for comparison, however I struggled to keep runtime live for long enough to complete the task. At this point, the other column of the data, the sentiment score, is translated to a 1 for positive or 0 for negative, with the lemmatized tokens appended afterward as a nested list. The writer wrote each row in that state as ppimdb.csv. Viewing the output, I had clearly introduced 1 too many '\n' characters in the for loop, however rather than attempting to preprocess the entire data set again, I filtered the data for lines consisting of only whitespace, creating a new csv, ppdata.csv. My data appeared to be sufficiently preprocessed at this point, pictured below.



I selected a Sigmoid activation function for task two, due to the output between 0 and 1. This function is ideal for binary classification. Sigmoid functions are prone to over saturation, which may be happening in this example. For comparison, it may be worth incorporating a Rectified Linear Activation Function as an alternative to Sigmoid. The neuron is fed an encoded array of tokens, as generated in step 1, and tokenized and padded and transformed to sequences by the Keras, which is scaled by NumPy into a range between 0 and 1. The product of inputs and weights is brought through the sigmoid function for the output of layer 1. Gradient, delta, and weighted delta are calculated to update the weights. Once the weights are updated the metrics including training loss and validation accuracy are recorded. This model appears to reach it's most accurate results between 100 and 200 epochs. While experimenting with learning rates, 0.5 was found to be effective. Higher learning rates appeared to lead to the model overfitting, as pictured below.

```
Epoch: 0, Training Loss: 0.29878280668612106, Validation Acc: 0.50016
Epoch: 100, Training Loss: 0.4653854844721281, Validation Acc: 0.49856
Epoch: 200, Training Loss: 0.44854202584841996, Validation Acc: 0.49696
Epoch: 300, Training Loss: 0.4547589020429763, Validation Acc: 0.49984
Epoch: 400, Training Loss: 0.4381567730624737, Validation Acc: 0.49592
Validation loss: 0.4805678454894797
Validation accuracy: 0.49736
```



The architecture for the multilayer perceptron for task 3 was similar to the single layer perceptron in task 2. A sigmoid activation function was again used. Weights are calculated twice, once for each of the layers. The layer 1 input and output are the same as they were in task 2, however in this model the output of layer 1 is fed into layer 2, and an array of sigmoid activated values is outputted. This model ran locally at about half the speed of running through Google Colab with a GPU, 8 minutes compared to 4.

For task 4 I chose a Logistic Regression classifier for comparison. While comparing validation loss and accuracy, perhaps unsurprisingly the multilayer perceptron consistently had the lowest validation loss, followed by Logistic Regression, followed closely by the single layer perceptron. Results of validation accuracy scores were much less consistent in terms of ranking, however they reliably fell within a smaller range (0.45-0.55).