

## CSE 574 Programming Assignment 2 Report

April 13th, 2020

Group 7: Nikhil Arora, Jacquelyn Dufresne, Mridula Rao

### Report 1: Word Features and Classification Model

The model implemented is a simple hand-crafted classifier that can label a movie review as positive or negative. After extracting the data files containing the reviews and their corresponding labels and splitting the reviews and labels into the train and test data, counter objects are used to store positive, negative, and all word counts for each training review. Every word in each training review is looped over and added to the counter that corresponds to the sentiment label of the review. The positive counter, `positive_word_count`, contains the word count of words in positive training reviews, the negative counter, `negative_word_count`, contains the word count of words in negative training reviews, and the counter `total_counts` contains the word count across all training reviews. Once these counters are created, a new counter, `pos_neg_ratios`, is created to hold the positive to negative ratios of each word. Words that appear more in positive training reviews have a ratio above 1, words that are neutral have a ratio of around 1, and words that appear more in negative training reviews have a ratio below 1. Only words with a count of over 100 are added to the `pos_neg_ratios` counter. Next, the ratios are converted to log values so that the positive values are close to 1, negative values are close to -1, and neutral values are around 0. Once the ratios are converted to log values, the word feature is complete. The completed word feature is then used to develop a classification model that is a rule-based system, assigning a positive or negative label to each test review. This classifier creates a new counter for each run, which stores the number of positive, negative, and neutral words in the provided test review. If the word has a log ratio greater than .5, the count of positive words is increased by 1. If the word has a log ratio less than -.5, the count of negative words is increased by 1. If the word has a log ratio between -.5 and .5, the count of neutral words is increased by 1. Once the entire review has been parsed, the classifier returns the test review as positive if the count of positive words in the review is greater than the count of negative words in

the review, and returns the review as negative if the count of negative words in the review is greater than the count of positive words in the review. This word feature and classification model produced an accuracy of .782 on the testing data, which can be seen in the screenshot below.

```
predictions_test = []
for r in reviews_test:
    l = nonml_classifier(r,pos_neg_ratios)
    predictions_test.append(l)

# calculate accuracy
correct = 0
for l,p in zip(sentiments_test,predictions_test):
    if l == p:
        correct = correct + 1
print('Accuracy of the model = {}'.format(correct/len(sentiments_test)))

Accuracy of the model = 0.782
```

## Report 2: Vanilla Neural Network

Note that the following snippet of code was included to get tensorflow to work for the upcoming tests for approach 2:

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
tf.compat.v1.random.set_random_seed(10)
```

The first test of the vanilla neural network was run on a laptop with a 7th generation Intel Core i5 processor, 8GB of RAM, and a Windows OS that was plugged in for the entirety of the run. The vanilla neural network included 50 epochs and one hidden layer with a width of 10. This produced a test accuracy of .847500 and a run time of approximately 63 seconds.

```
Train acc: 0.945708, Test_acc: 0.847500  
Time elapsed - 63.06840395927429 seconds.
```

To check the relevance of a faster machine, we also ran the model in Google Colab. Below is the run time for comparisons. The run in Google Colab produced a test accuracy of .85750 and a much faster run time of approximately 50 seconds. These two tests have similar accuracies, however Google Colab was able to run the mode approximately 16 seconds faster.

```
Train acc: 0.933750, Test_acc: 0.857500  
Time elapsed - 47.76960349082947 seconds.
```

---

The accuracy looked to be more realistic on the vanilla neural network as the difference in the training accuracy and test accuracy is very minimal. Also, both runs of the vanilla neural network produced a much higher accuracy than the rule-based algorithm implemented in approach 1, which produced an accuracy of .782. Using a model that produces an accuracy of between .84-.85 is significantly better than a model that only produces an accuracy of .782 and has static results.

### Report 3: Improved Neural Network

It was very interesting to see the difference in the outputs of the neural network depending on the epoch values, number of hidden layers, and the width of the layers. We experimented with the change in output by changing only the epoch values, the number of hidden layers, and the width of the hidden layers. Note that all tests in this section were run using Google Colab for the sake of consistency. Below are different trials using different values for the number of epochs, number of hidden layers, and the width of the hidden layers.

Trial 1 :-

Epoch = 50

Hidden layers = 2

Width of layer 1 = 10

Width of layer 2 = 10

```
Train acc: 0.905125, Test_acc: 0.837500  
Time elapsed - 40.2413592338562 seconds.
```

Using only two hidden layers with a width of 10 and 50 epochs, the accuracy of the neural network was .83750, which is only slightly lower than the .84-.85 accuracy range of the vanilla neural network. Adding a second hidden layer does not appear to affect the accuracy much when using the original 50 epochs and hidden layer width of 10.

Trial 2 :-

Epoch = 100

Hidden layers = 2

Width of layer 1 = 10

Width of layer 2 = 10

```
Train acc: 0.922958, Test_acc: 0.830000  
Train acc: 0.924708, Test_acc: 0.825000  
Time elapsed - 92.87957811355591 seconds.
```

Changing the number of epochs to 100 and keeping the number of hidden layers at 2 with a width of 10 produced a slightly lower accuracy of .8250 than the previous accuracy of .83750 of the same model with 50 epochs. The run time was also significantly longer after increasing the number of epochs from 50 to 100, increasing from approximately 40 to 92 seconds.

Trial 3:-

Epoch = 1

Hidden layers = 2

Width of layer 1 = 10

Width of layer 2 = 10

```
Train acc: 0.500958, Test_acc: 0.505000  
Time elapsed - 2.6236748695373535 seconds.
```

---

When using two hidden layers with a width of 10 and decreasing the number of epochs to one, the test accuracy plummeted to .5050. The run time was extremely fast, only taking approximately 2.6 seconds. Despite the speed of this variation of the neural network, the accuracy makes only using one epoch useless.

Trial 4 :-

Epoch = 50

Hidden layers = 2

Width of layer 1 = 15

Width of layer 2 = 10

```
Train acc: 0.887292, Test_acc: 0.852500  
Time elapsed - 42.95487189292908 seconds.
```

---

Using an epoch value of 50, two hidden layers, and increasing the width of the first hidden layer to 15 and using a width of 10 for the second hidden layer, the model produced a desirable accuracy of .85250. This is slightly better than the results produced by the vanilla neural network. The run time was also two seconds slower than the variation of the neural network that had an epoch of 50, two hidden layers, and a width of 10 for both layers. Increasing the width of the first layer to 15 made the model run slightly slower.

Trial 5 :-

Epoch = 75

Hidden layers = 2

Width of layer 1 = 25

Width of layer 2 = 13

```
Train acc: 0.876500, Test_acc: 0.820000  
Time elapsed - 63.11441946029663 seconds.
```

---

Increasing the number of epochs from 50 to 75, using two hidden layers, and using the widths 25 and 13 for the two hidden layers produced an accuracy of .82, which is lower than the accuracy of the vanilla neural network and the accuracy. The run time was also significantly longer, where the vanilla network took approximately 47 seconds to run, while this variation of the neural network took approximately 63 seconds to run.

Trial 6 :-

Epoch = 50

Hidden layers = 3

Width of layer 1 = 10

Width of layer 2 = 10

Width of layer 3 = 10

```
Train acc: 0.851583, Test_acc: 0.827500  
Time elapsed - 41.540313959121704 seconds.
```

---

Adding many layers can often result in the overfitting of data. However after increasing the number of hidden layers to three while keeping the number of epochs at 50 and the width of the hidden layers at 10, there was only a minimal decrease in the accuracy of the test data. The accuracy decreased from the .84-.85 range to .827500 when using 3 hidden layers with a width of 10 and 50 epochs.

Trial 7 :-

Epoch = 50

Hidden layers = 3

Width of layer 1 = 15

Width of layer 2 = 10

Width of layer 3 = 10

```
Train acc: 0.849458, Test_acc: 0.827500  
Time elapsed - 68.53975176811218 seconds.
```

---

Keeping the neural network with three hidden layers and 50 epochs, the width of the first hidden layer was increased to 15, and the width of the second and third hidden layer was kept at 10. The accuracy happened to be identical as the previous trial, however the run time got significantly slower, increasing by a little under 30 seconds.

The accuracy value of the model with and without using the function **find ignore words** are as below for vanilla neural network with one hidden layer :-

Trial 8 :-

Epoch = 50

Hidden layers = 1

Width of layer 1 = 10

a) Accuracy without the function **find ignore words** :-

```
Train acc: 0.933750, Test_acc: 0.857500  
Time elapsed - 47.76960349082947 seconds.
```

---

b) Accuracy with the function **find ignore words** :-

```
Train acc: 0.938750, Test_acc: 0.862500  
Train acc: 0.939417, Test_acc: 0.836250  
Time elapsed - 46.19692611694336 seconds.
```

---

When using a set of ignored words on the vanilla neural network, the run time decreased by approximately one second. Although this does not appear to be a significant decrease in run time, having a slightly faster model in this vanilla neural network may translate to significantly faster

run times when increasing the number of hidden layers and epochs by a large number.

Overall, it appears the number of epochs, number of hidden layers, and the width of the hidden layers can impact the accuracy of the neural network. In this instance, it appears the number of epochs had the most impact on the neural network, as having one epoch made the accuracy poor, and having too many epoch's made the accuracy and run time both poor. Increasing the number of hidden layers can also have a significant impact on the accuracy, as the model will overfit to the train data. Variations of the neural network must be played with to find the optimal model to produce the best accuracy on the test data.

## Report 4: Image Classification Neural Network

### 1. Evaluating the Hidden Layer Neural Network Provided:

Using a laptop with a 7th generation Intel Core i5 processor, 8GB of RAM, and a Windows OS that was plugged in for the entirety of the run, the running time of each epoch ran in the beginning around 15 and 16 seconds, and sped up to about 14 seconds for a majority of the run time. The accuracy of the neural network started at .6129 and increased to the .79 range after approximately 100 runs and finished with an accuracy of .7954, which can be seen in the screenshot below.

```
Epoch 500/500  
100000/100000 [=====] - 14s 137us/step - loss: 0.7865 - accuracy: 0.7954
```

### 2. Comparing the performance when the number of hidden layers are increased to 3 and 5:

Using a laptop with a 7th generation Intel Core i5 processor, 8GB of RAM, and a Windows OS that was plugged in for the entirety of the run, the number of hidden layers were increased to three and five for evaluation. For three hidden layers, the running time of each epoch typically fell between 19 and 22 seconds. During the first 6 epochs of the neural network



with three hidden layers, the accuracy of the test data increased with each run. However, after the 7th epoch, the accuracy of the test data began to decrease. There was a significant dip in accuracy of the data after the 9th run, where the accuracy of the test data fell from .6502 to .4500, .4500 to .3313, and so on. This can be seen in the screenshot below.

```
Epoch 1/500
100000/100000 [=====] - 22s 216us/step - loss: 1.5141 - accuracy: 0.6767
Epoch 2/500
100000/100000 [=====] - 21s 212us/step - loss: 0.9491 - accuracy: 0.7516
Epoch 3/500
100000/100000 [=====] - 21s 210us/step - loss: 0.9713 - accuracy: 0.7620
Epoch 4/500
100000/100000 [=====] - 20s 203us/step - loss: 1.0119 - accuracy: 0.7727
Epoch 5/500
100000/100000 [=====] - 20s 202us/step - loss: 1.0064 - accuracy: 0.7784
Epoch 6/500
100000/100000 [=====] - 21s 211us/step - loss: 1.0339 - accuracy: 0.7798
Epoch 7/500
100000/100000 [=====] - 21s 208us/step - loss: 1.0549 - accuracy: 0.7661
Epoch 8/500
100000/100000 [=====] - 21s 211us/step - loss: 1.1768 - accuracy: 0.7302
Epoch 9/500
100000/100000 [=====] - 25s 249us/step - loss: 1.4870 - accuracy: 0.6502
Epoch 10/500
100000/100000 [=====] - 26s 257us/step - loss: 2.2290 - accuracy: 0.4500
Epoch 11/500
100000/100000 [=====] - 22s 220us/step - loss: 2.7290 - accuracy: 0.3313
Epoch 12/500
100000/100000 [=====] - 22s 216us/step - loss: 3.0812 - accuracy: 0.2873
Epoch 13/500
100000/100000 [=====] - 20s 197us/step - loss: 3.4701 - accuracy: 0.2515
Epoch 14/500
100000/100000 [=====] - 20s 201us/step - loss: 3.7932 - accuracy: 0.2421
Epoch 15/500
100000/100000 [=====] - 20s 199us/step - loss: 2.8720 - accuracy: 0.2002
```

The final epoch of the three layer neural network produced a very low accuracy. The accuracy is incredibly poor compared to the neural network with one hidden layer, which finished with an accuracy of .7954 when ran on the laptop with a 7th generation Intel Core i5 processor, 8GB of RAM, and a Windows OS. A screenshot of the final accuracy can be seen below.

```
Epoch 500/500
100000/100000 [=====] - 19s 194us/step - loss: 2.4514 - accuracy: 0.2113
```

When increasing the number of hidden layers from 3 to 5, a similar process occurred. For five hidden layers, the running time of each epoch typically fell between 23 and 26 seconds. During the first 3 epochs of the neural network with five hidden layers, the accuracy of the test data increased with each run. However, after the 4th epoch, the accuracy of the test data began to decrease. There was a significant dip in accuracy of the data after the 10th run, where the accuracy of the test data fell from .7036 to .5902, .5902 to .3292, and so on. This can be seen in

the screenshot below.

```
Epoch 1/500
100000/100000 [=====] - 26s 259us/step - loss: 1.1206 - accuracy: 0.6957
Epoch 2/500
100000/100000 [=====] - 26s 260us/step - loss: 0.8673 - accuracy: 0.7738
Epoch 3/500
100000/100000 [=====] - 25s 249us/step - loss: 0.8447 - accuracy: 0.7888
Epoch 4/500
100000/100000 [=====] - 26s 257us/step - loss: 0.9593 - accuracy: 0.7675
Epoch 5/500
100000/100000 [=====] - 26s 258us/step - loss: 1.0938 - accuracy: 0.7308
Epoch 6/500
100000/100000 [=====] - 25s 251us/step - loss: 1.3596 - accuracy: 0.6955
Epoch 7/500
100000/100000 [=====] - 25s 253us/step - loss: 1.2779 - accuracy: 0.7221
Epoch 8/500
100000/100000 [=====] - 25s 249us/step - loss: 1.1432 - accuracy: 0.7323
Epoch 9/500
100000/100000 [=====] - 26s 257us/step - loss: 1.2471 - accuracy: 0.7379
Epoch 10/500
100000/100000 [=====] - 25s 252us/step - loss: 1.5268 - accuracy: 0.7036
Epoch 11/500
100000/100000 [=====] - 26s 258us/step - loss: 1.9091 - accuracy: 0.5902
Epoch 12/500
100000/100000 [=====] - 26s 256us/step - loss: 2.3880 - accuracy: 0.3292
Epoch 13/500
100000/100000 [=====] - 25s 254us/step - loss: 2.7241 - accuracy: 0.2816
Epoch 14/500
100000/100000 [=====] - 26s 261us/step - loss: 2.9895 - accuracy: 0.2575
Epoch 15/500
100000/100000 [=====] - 26s 256us/step - loss: 2.7875 - accuracy: 0.2139
```

The final accuracy of .1005 is incredibly poor compared to the neural network with one hidden layer, which finished with an accuracy of .7954, and even worse than the already poor accuracy of the neural network with three hidden layers, which finished with an accuracy of .2113. The run time of the model also become longer as the number of hidden layers was increased, where the run time for the neural network with one hidden layer ranged between 14 and 16 seconds per epoch, the run time for the neural network with three hidden layers ranged between 19 and 22 seconds per epoch, and the run time for the neural network with five hidden layers ranged between 23 and 26 seconds per epoch. Keep in mind these three comparisons are regarding runs on the laptop with a 7th generation Intel Core i5 processor, 8GB of RAM, and a Windows OS. A screenshot of the final accuracy can be seen below.

```
Epoch 500/500
100000/100000 [=====] - 23s 233us/step - loss: 2.3027 - accuracy: 0.1005
```

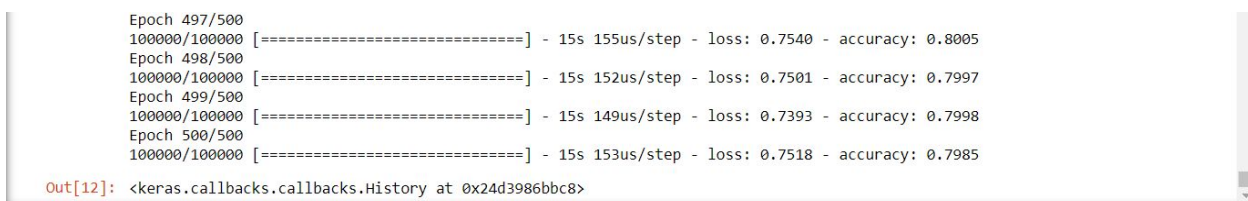
It is likely that adding more layers to the network made it too complex, and the model was overfitting the training data. This is a great illustration of the impact of overfitting can have on the accuracy of the model on the training data. Oftentimes, the simpler model produces better results.

3. Use the `resize images()` function to reduce the resolution of the images to  $(20 \times 20)$ ,  $(15 \times 15)$ ,  $(10 \times 10)$  and  $(5 \times 5)$ . Using the 1 hidden layer architecture, compare the performance at different resolutions, including the original  $(28 \times 28)$  resolution, both in terms of test accuracy and time

Using a laptop with a 8th generation Intel Core i7 processor, 16GB of RAM, and a Windows OS, the `resize images()` function was used to reduce the resolution of the images and compare performance at different resolutions.

### For 20 x 20 resolution:

It took about 24 seconds to run in the beginning for each epoch. However after a few epochs, run time increased to 15 seconds on average. Also, the accuracy started at 0.5952 but reached 0.7985 after 500 epochs, as you can see the output in the screenshot below:



```
Epoch 497/500
100000/100000 [=====] - 15s 155us/step - loss: 0.7540 - accuracy: 0.8005
Epoch 498/500
100000/100000 [=====] - 15s 152us/step - loss: 0.7501 - accuracy: 0.7997
Epoch 499/500
100000/100000 [=====] - 15s 149us/step - loss: 0.7393 - accuracy: 0.7998
Epoch 500/500
100000/100000 [=====] - 15s 153us/step - loss: 0.7518 - accuracy: 0.7985
Out[12]: <keras.callbacks.callbacks.History at 0x24d3986bbc8>
```

### For 15 x 15 resolution:

It took about 6 seconds to run in the beginning for each epoch but after a few epochs, run time increased to 15 seconds on average. Also, the accuracy started at 0.6035, but increased to .7810

after 500 epochs, as you can see the output in the screenshot below:

```
100000/100000 [=====] - 15s 149us/step - loss: 0.8224 - accuracy: 0.7812
Epoch 497/500
100000/100000 [=====] - 15s 146us/step - loss: 0.8431 - accuracy: 0.7806
Epoch 498/500
100000/100000 [=====] - 15s 151us/step - loss: 0.8202 - accuracy: 0.7817
Epoch 499/500
100000/100000 [=====] - 15s 149us/step - loss: 0.8199 - accuracy: 0.7815
Epoch 500/500
100000/100000 [=====] - 15s 149us/step - loss: 0.8351 - accuracy: 0.7810

Out[29]: <keras.callbacks.callbacks.History at 0x291c4de0d08>
```

### For 10 x 10 resolution:

It took about 16 seconds in the starting for each epoch but after a few epochs time reduced to 14 seconds. Also, the accuracy with which it started was 0.6006 but after 500 epochs it reached 0.8097 as you can see the output in the screenshot below

```
Epoch 496/500
100000/100000 [=====] - 14s 143us/step - loss: 0.7326 - accuracy: 0.8101
Epoch 497/500
100000/100000 [=====] - 14s 144us/step - loss: 0.7377 - accuracy: 0.8106
Epoch 498/500
100000/100000 [=====] - 14s 143us/step - loss: 0.7300 - accuracy: 0.8118
Epoch 499/500
100000/100000 [=====] - 14s 141us/step - loss: 0.7311 - accuracy: 0.8102
Epoch 500/500
100000/100000 [=====] - 14s 142us/step - loss: 0.7296 - accuracy: 0.8097

Out[16]: <keras.callbacks.callbacks.History at 0x291c046bc48>
```

### For 5 x 5 resolution:

It took about 6 seconds in the starting for each epoch but after a few epochs time increased to 15 seconds on average. Also the accuracy with which it started was 0.6054 but after 500 epochs it reached 0.8043 as you can see the output in the screenshot below

```
Epoch 497/500
100000/100000 [=====] - 15s 146us/step - loss: 0.7403 - accuracy: 0.8047
Epoch 498/500
100000/100000 [=====] - 14s 145us/step - loss: 0.7397 - accuracy: 0.8044
Epoch 499/500
100000/100000 [=====] - 15s 147us/step - loss: 0.7302 - accuracy: 0.8046
Epoch 500/500
100000/100000 [=====] - 15s 150us/step - loss: 0.7281 - accuracy: 0.8043

Out[23]: <keras.callbacks.callbacks.History at 0x291c1a2a508>
```