COMP 4107 Assignment Three:

Connor Raymond Stewart, 101041125

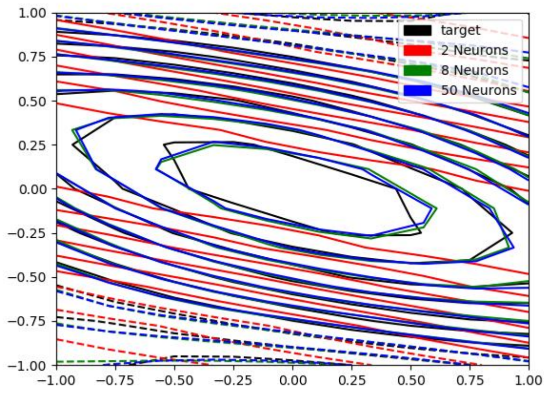
Gabrielle Latreille, 101073284

Note: Code was tested in Python 3.7.4 using TensorFlow 2.4.1 with NumPy 1.19.5

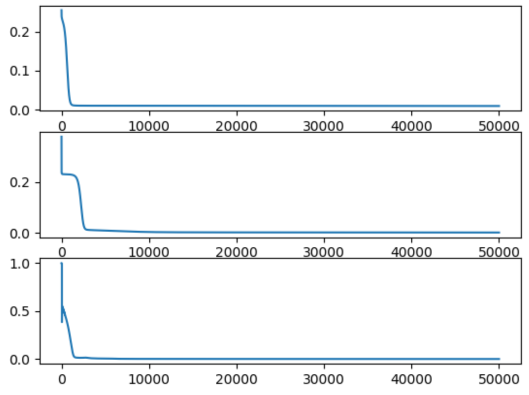
Q1: The graphs and tables can be reproduced using the q1.py file. Please note that since all datasets are random, there is a chance that a dataset wont output results exactly like the test cases shown below. Simply rerunning the code until you see the desired output can work.

a) The grid with dimensions ten squared as described in the assignment was randomized with random ordering and number generation. The values for the function were generated and stored in a matrix of type [x, y, f(x,y)] for all the training items. There was a total of 100 training items, a ten-by-ten grid, fifty-thousand epochs, and a test set of size nine by nine. The error prediction values were overall low for all the neuron tests, but notably so for the eight and fifty neuron models.

Contour Mapping:



The mean square error cost for each epoch:

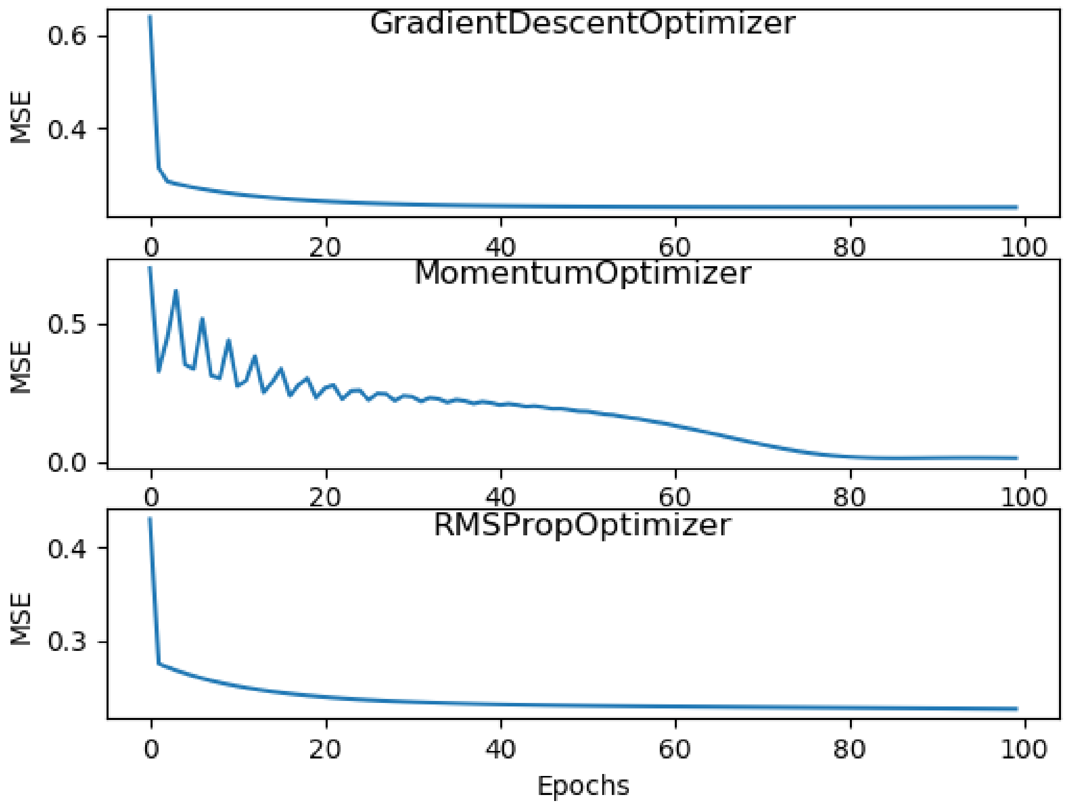


When using the eight-neuron model for the hidden layer in the neural network, we see that the MSE is much very close to the fifty-layer network. This indicates the eight-neuron network could be better since it is more efficient and has a lower chance of overfitting. For this part, the learning rate was set to 0.1 and the epochs 50000 as was required. See the attached terminal output below:

```
The mean square error of the following networks is:
               2 neurons:  0.46299194067822047
               8 neurons:  0.4712547931249993
              50 neurons:  0.47124251996106814
```

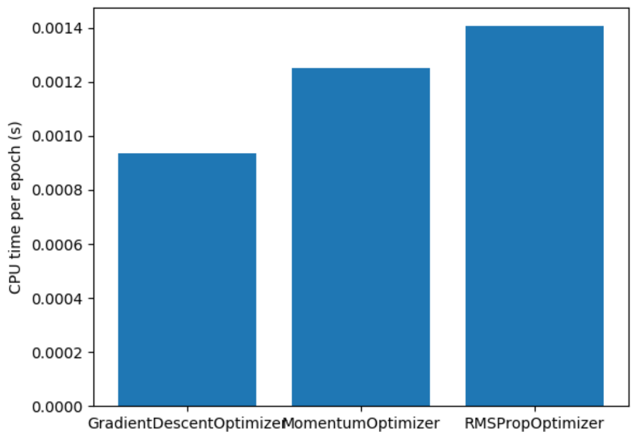
b) As required per the problem, the tolerances for the MSE are set to 0.02 for the training data, and the number of epochs and convergence are set to 100. It is clearly visible that the RMSPropOptimizer converges the fastest, followed by the gradient descent optimizer. The momentumOptimizer does not converge very fast but this is because it is a unoptimized, unlike the RMSPropOptimizer.

<u>Convergence per Optimizer:</u>



As can be seen, the slower computation time per epoch in the RMSPropOptimizer is vastly made by for by its convergence rate when compared to the gradient descent optimizer. For very complex problems requiring high precision, the RMS tool is probably better, and it compensates for slower run time by requiring less epochs (due it its fast convergence). For problems needing less precision, the gradient descent optimizer could be better, since its initial convergence (as seen above) is comparable to the RMS tool, however the trade off is that it is less precise over more epochs. Thus, when runtime is concerned, the gradient descent tool is better, otherwise the RMS tool is the best:

<u>Runtime Per Epoch by Optimization Tool Used:</u>



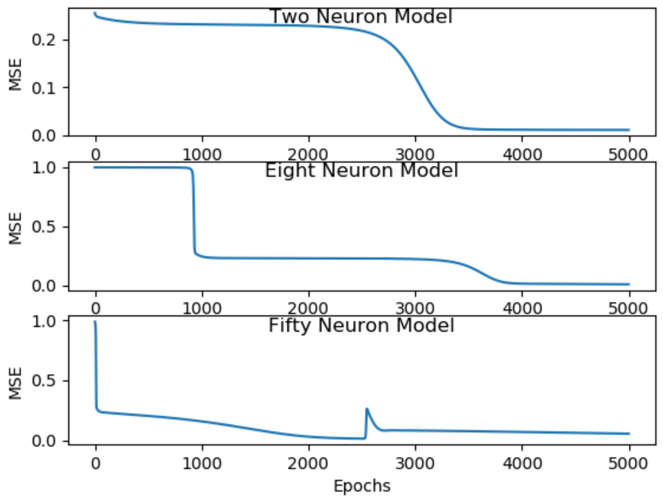
When checking the MSE rate for each optimizer at the 100th epoch, we notice another interesting trend. Although the RMSPropOptimizer tool converges fast, it is not necessarily more accurate. The momentumOptimizer has the lowest MSE and reaches its threshold much faster than the other tools. The gradient descent optimizer has a higher error then the other two tools and takes many more epochs to reach its error threshold. See the attached terminal output below:

```
MSE for optimizers with a learningRate of 0.02 and 100 epochs:
Cost for GradientDescentOptimizer: 0.23113462
Cost for MomentumOptimizer: 0.014536987
Cost for RMSPropOptimizer: 0.22652644
MSE and iteration for learningRate of 0.02 when stopping on training error less then 0.02:
Cost for GradientDescentOptimizer: 0.01997649 after 806
Cost for MomentumOptimizer: 0.01850546 after 113
Cost for RMSPropOptimizer: 0.019876195 after 473
```

All in all, we notice that the gradient descent optimizer is the fastest per epoch but is slow to converge and has a higher MSE per epoch taken. The momentumOptimizer has the lowest MSE per epoch and reaches its threshold the fastest, meaning it is more accurate, however it converges slower then the RMSPropOptimizer. The RMSPropOptimizer converges quite quickly at the expense of a slower execution per epoch and MSE, but it converges quick enough that it is the best tool. Overall, the best tool depends on the goal of the neural network, but for this the RMSPropOptimizer works best.

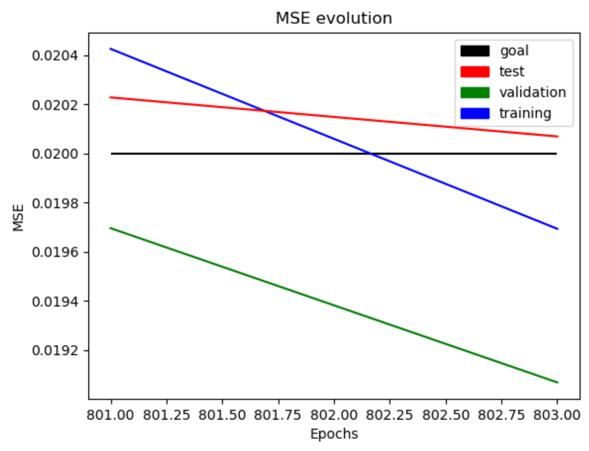
c) We can verify that the eight-neuron model is the best choice since we see that early stopping does nothing for it, indicating the model is fitting well. Although the fifty-neuron model has no early stopping, we note that sometimes the MSE spikes for a few epochs. It is likely the case that the fifty-neuron model tends to overfit, causing the MSE to occasionally spike, meaning that it has more neurons then is optimal for the model. It is also visible that the fifty-neuron model is not converging correctly, and that it almost plateaus between 3000 and 5000 epochs since the MSE is decreasing so slowly. See below, that the two-neuron model converges correctly - but is inaccurate due to underfitting - and that the fifty-neuron model has a spike in its MSE indicating overfitting. Also note that the eight-neuron model actually converges without any spikes:

Convergence rates for MSE by Neuron Model:



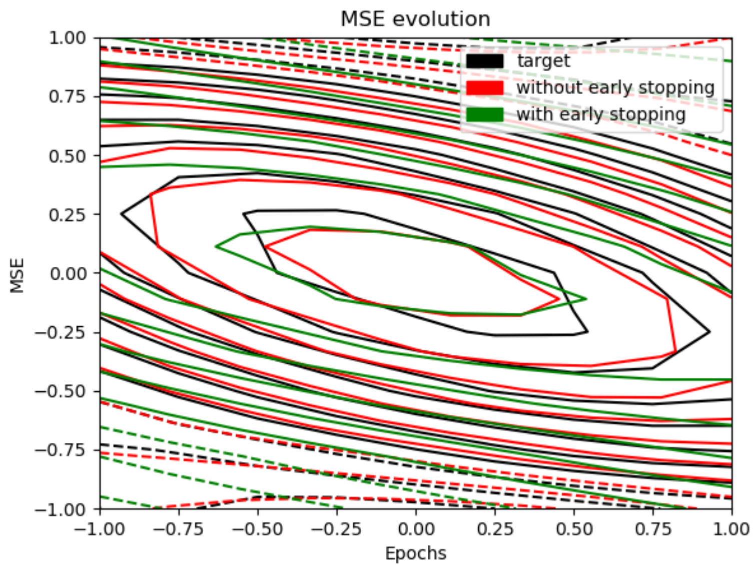
The MSE values cross the threshold at epoch 802. This pattern is what we would logically expect to see. The training set crosses the threshold past the threshold before the testing set (likely some mild overfitting) and all three sets continue to decease to the next epoch past 802. The validation data is a random set for comparison, and it is distinct from the training data, but follows the same trend. Lastly, we see that the variance in error increases as the training and testing errors start to diverge past 802.

MSE Evolution per Parameter:



As can be seen below, early stopping for the contour mapping makes little overall difference to the final diagram, but it does provide the ability to shorten the runtime to yield a very similar contour map. However, the random elements of the program such as the initial weights and orderings can heavily modify the below diagram. On some runs of the program, early stopping sometimes results in a contour map which is strongly divergent from the contour map without stopping. Thus, we can conclude that early stopping gives a user the chance to shorten runtimes, but it must be implemented with perfect error margins and other parameters to prevent an over or underfitted model from prematurely being outputted.

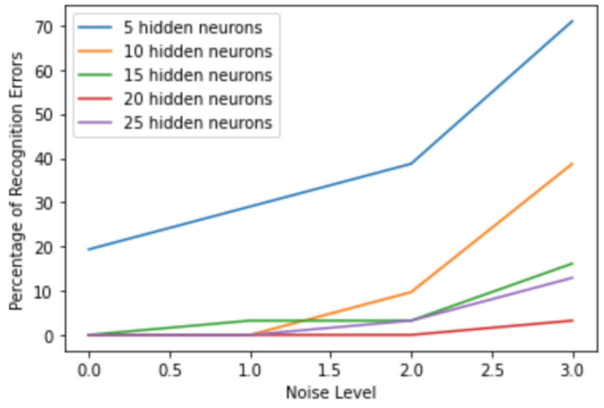
MSE With and Without Early Stopping:



Q2: To get the following results run the Q2 python files in the submission. The program executes multiple neural-network models with verifying numbers of neurons for the hidden layer. All models are executed with their noise levels recorded and plotted into the graph below. The networks were run using a learning rate of 0.01, which resulted in a quick convergence for the program. All networks in part a ran for roughly 100 epochs or less:

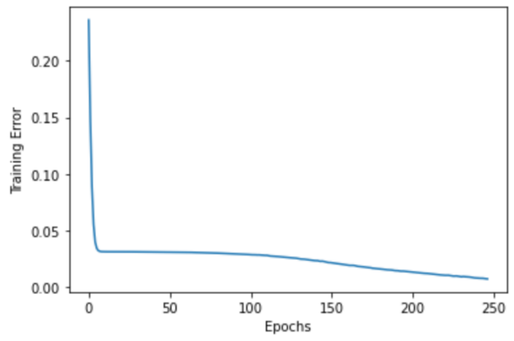
a) Code in q2a.ipynb

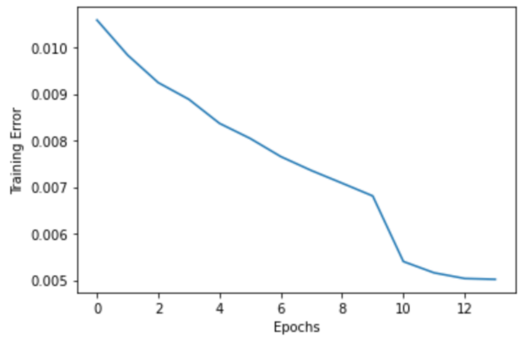
<u>Noise Level by Hidden Neurons in Neural-Network:</u>



b) Code in q2b.ipynb
The number of hidden neurons used for the below models is twenty, as it resulted in the best accuracy and lowest noise levels. The convergence rate of the twenty-neuron model is also descent which helped it get chosen. In the images below, the first image is based on the original data provided, and the second image (which is more vertical) is trained with the noisy data for ten epochs of the network.

<u>Training Error over Epochs:</u>

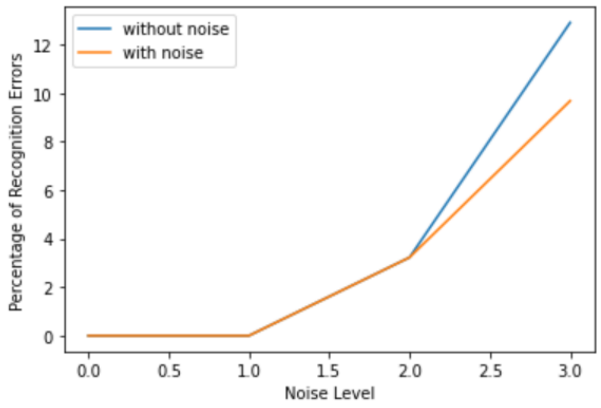




c) Code in q2c.ipynb

The network was trained with both the original and noisy data to find convergence. The original data actually had more recognition errors with greater noise then the noisy data. Since the noisy data tests on more data, it likely has a larger range of valid recognitions which lowers its recognition errors. This larger range is of course at the expense of the increased noise. The trade-off between noisy data and a larger recognition range can be seen below:

<u>Recognition Error by Noise Level:</u>


without noise
with noise
Percentage of Recognition Errors
Noise Level