Assignment 3

COMP 4107– Assignment 3

Due: Thursday, 11 Mar 2021

Author: Group 29: Devon Robitaille, Alex Fillips

Carleton University

# Question 1 – Completed by Devon Robitaille

## Part (a) – Network Design

1. Reference supplied python code to see how *GradientDescentOptimizer* was implemented.

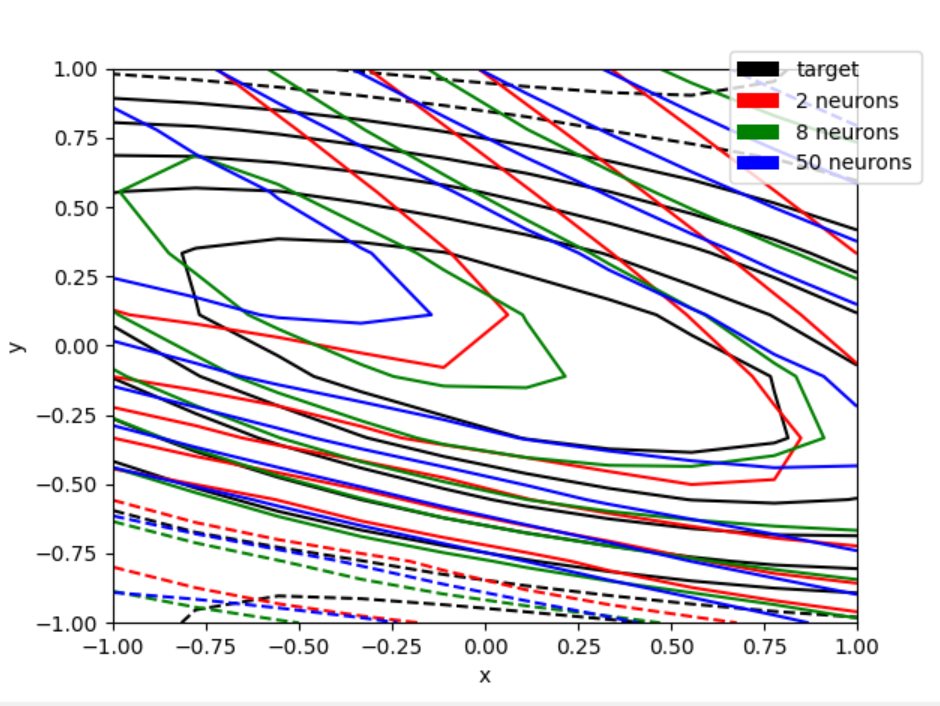


Figure 1- Function Contours

|  |  |  |
| --- | --- | --- |
| **# of Neurons** | **MSE / Loss** | **Epochs** |
| 2 | 0.08053534477949142 | 2289 |
| 8 | 0.07150878012180328 | 434 |
| 50 | 0.05533287301659584 | 932 |

1. For our experiments we used the “*tanh*” function as part of the TensorFlow library for the activation function of the hidden layers.

## Part (b) – Training

1. I set the total number of epochs to 2000 for this question to see at which point the optimizers would converge towards MSE of 0.02.

|  |  |
| --- | --- |
| **Model** | **Epochs to convergence** |
| Gradient Descent | 311 |
| Momentum | (Started at 0.001) Therefore, 0 |
| RMSProp | 40 |

1. MSE against epoch

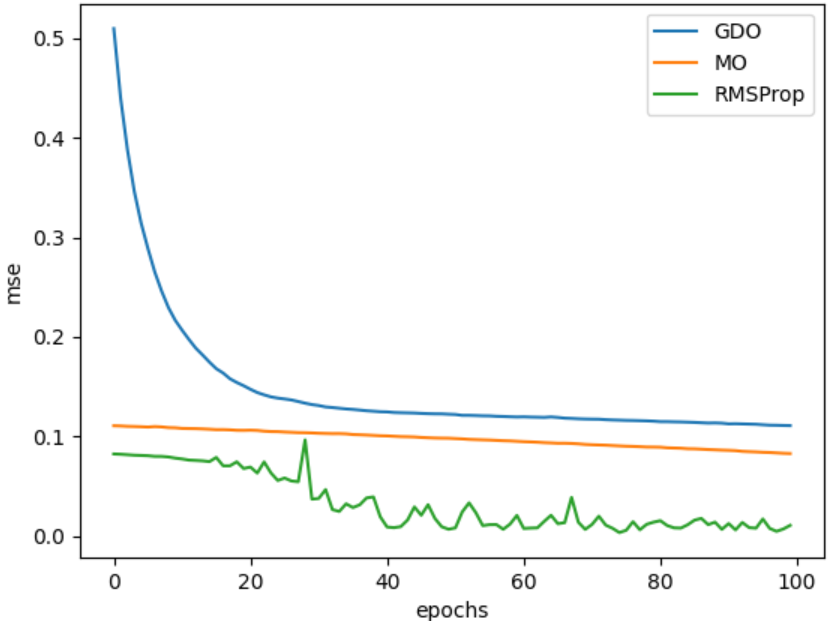


Figure 2 - MSE against epoch number

1. Bar graph cpu time against 3 methods

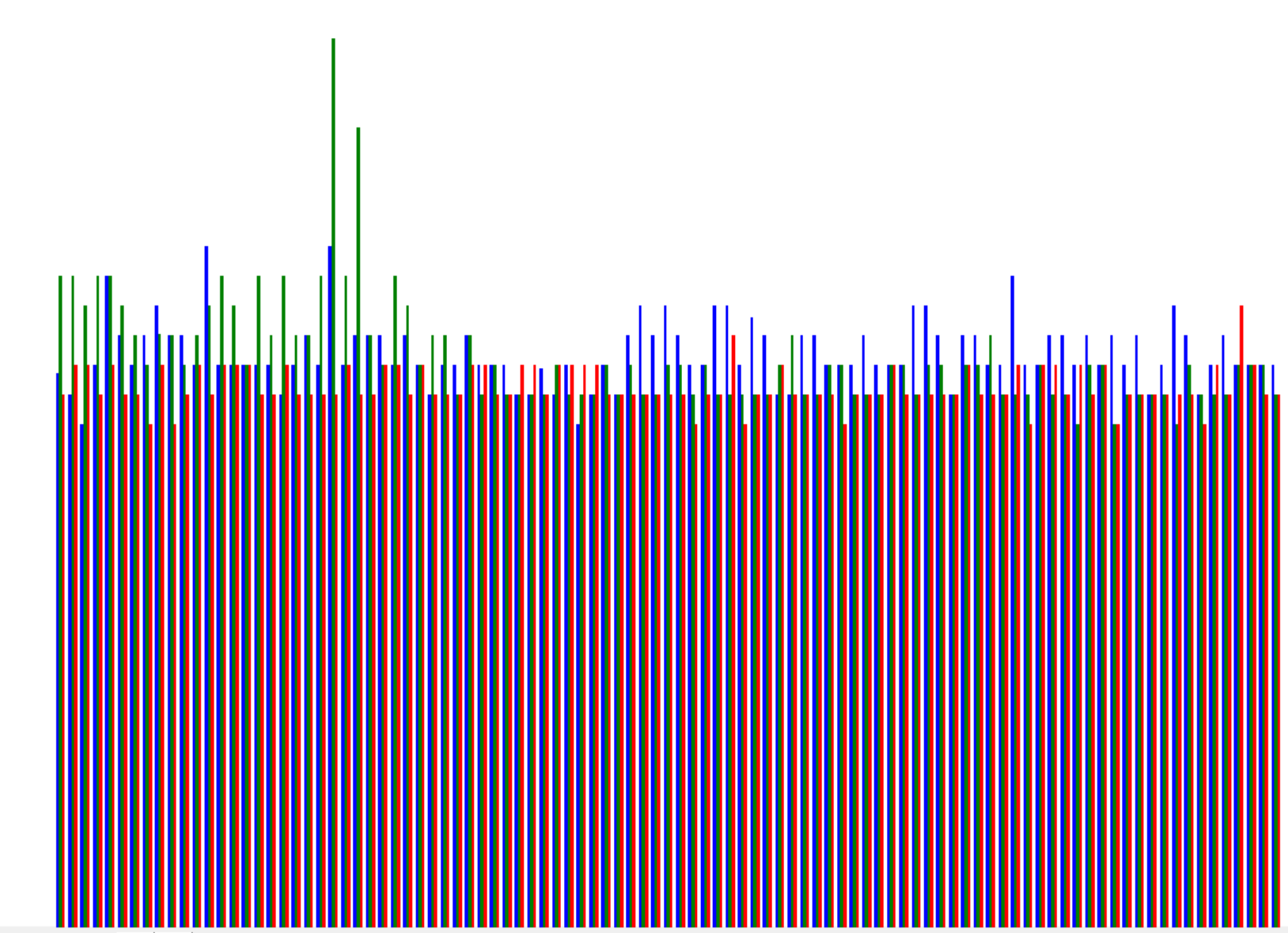


Figure 3 - CPU time against 3 methods

1. Best accuracy at the end of 100 epochs

Based on the supplied *Fig.2* supplied in question 2, we can see that as the optimizers approached epoch 100, RMSProp provided the highest level of accuracy which can be visualized by the amount of loss or MSE.

1. Most accurate when the training error is reached

|  |  |
| --- | --- |
| **Model** | **Accuracy at end of training** |
| Gradient Descent | 0.06798079 |
| Momentum | 0.06798079 |
| RMSProp | 0.06798079 |

Since all three models provided the same level of accuracy by taking the average of the prediction against a range of possible values. It would say that they are all equally accurate.

## Part (c) – Early Stopping

1. Confirm that approximately 8 neurons are a good choice for the current problem

Using all of the previous data that has been calculated along with the interpretation taken from the different plots, we can say that using approximately 8 neurons appears to be the best choice for the current problem. By using 8 hidden neurons of the neural network, we can get the best mix of total number of epochs to train against the accuracy of the network which can be measured in MSE. Mainly that we were able to reach the desired MSE threshold of 0.02 most consistently which 8 neurons. Following this, we can see from the graph supplied under “run experiments” *Fig.4* that having 8 neurons seems to fit around average while also reaching this state a lot sooner than its counter parts.

1. Run experiments…

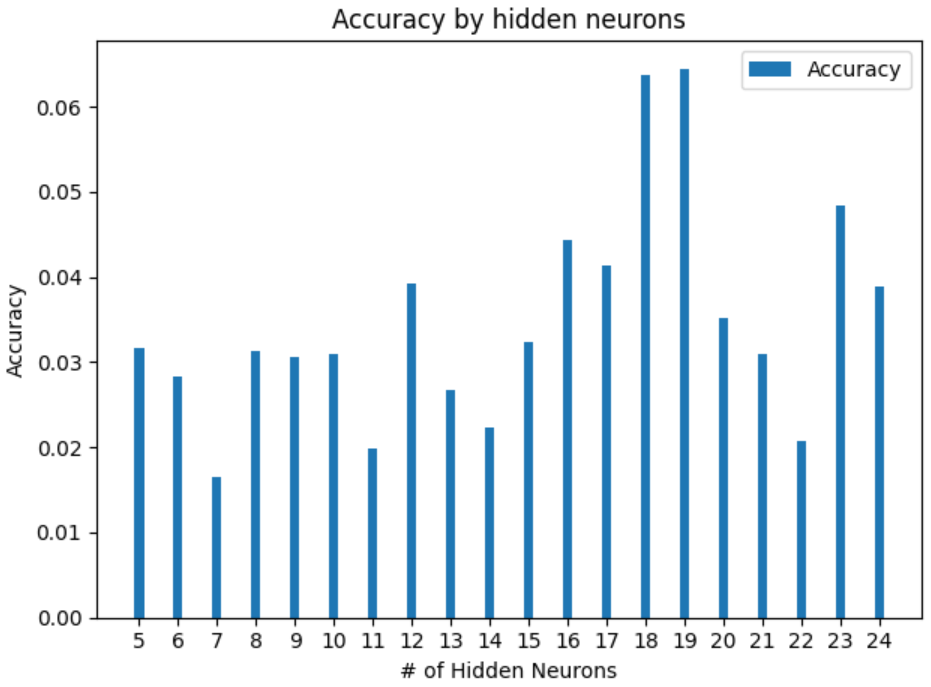


Figure 4 - Evolution of Accuracy for 500 Epochs

1. Figure 6 and 7

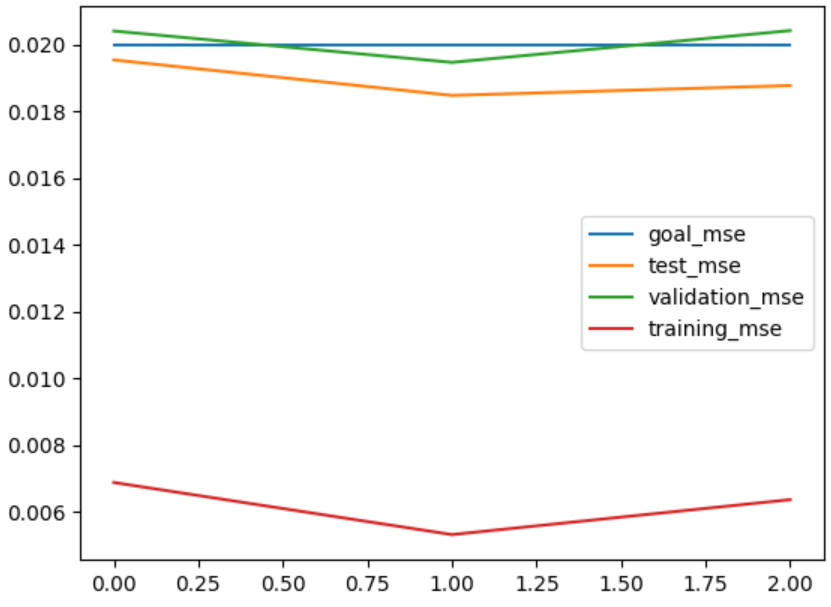


Figure 5 - Evolution of the MSE

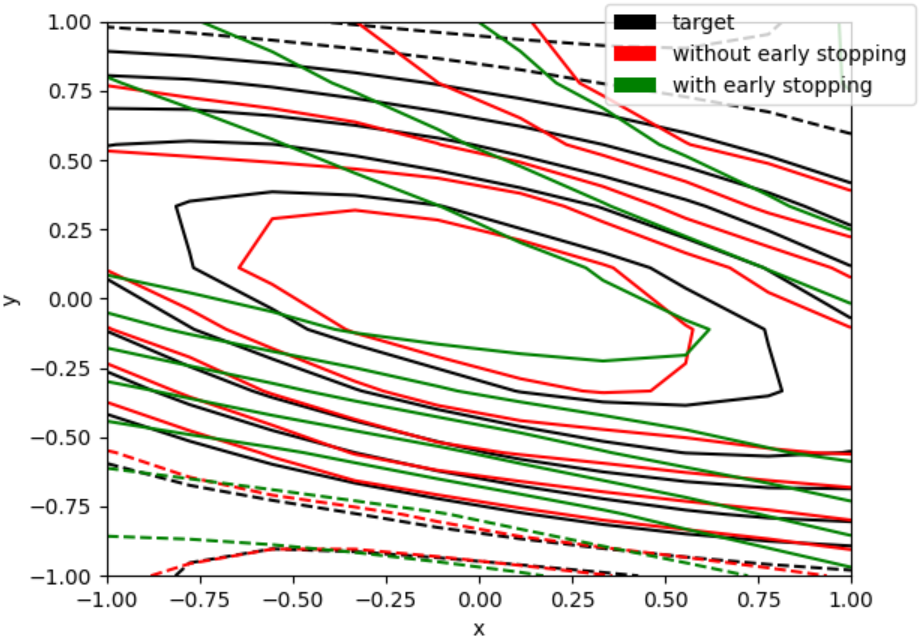


Figure 6- Function Contours

# Question 2 – Completed by Alex Fillips

## Part (a) – Network Design

1. Run experiments with hidden neuron numbers in the range 5-25.

See Q2PartA.py and model\_Q2PartA.py to see the creation and training of networks with varying sizes of hidden layers.

1. Plot a chart of recognition error against the number of hidden neurons.

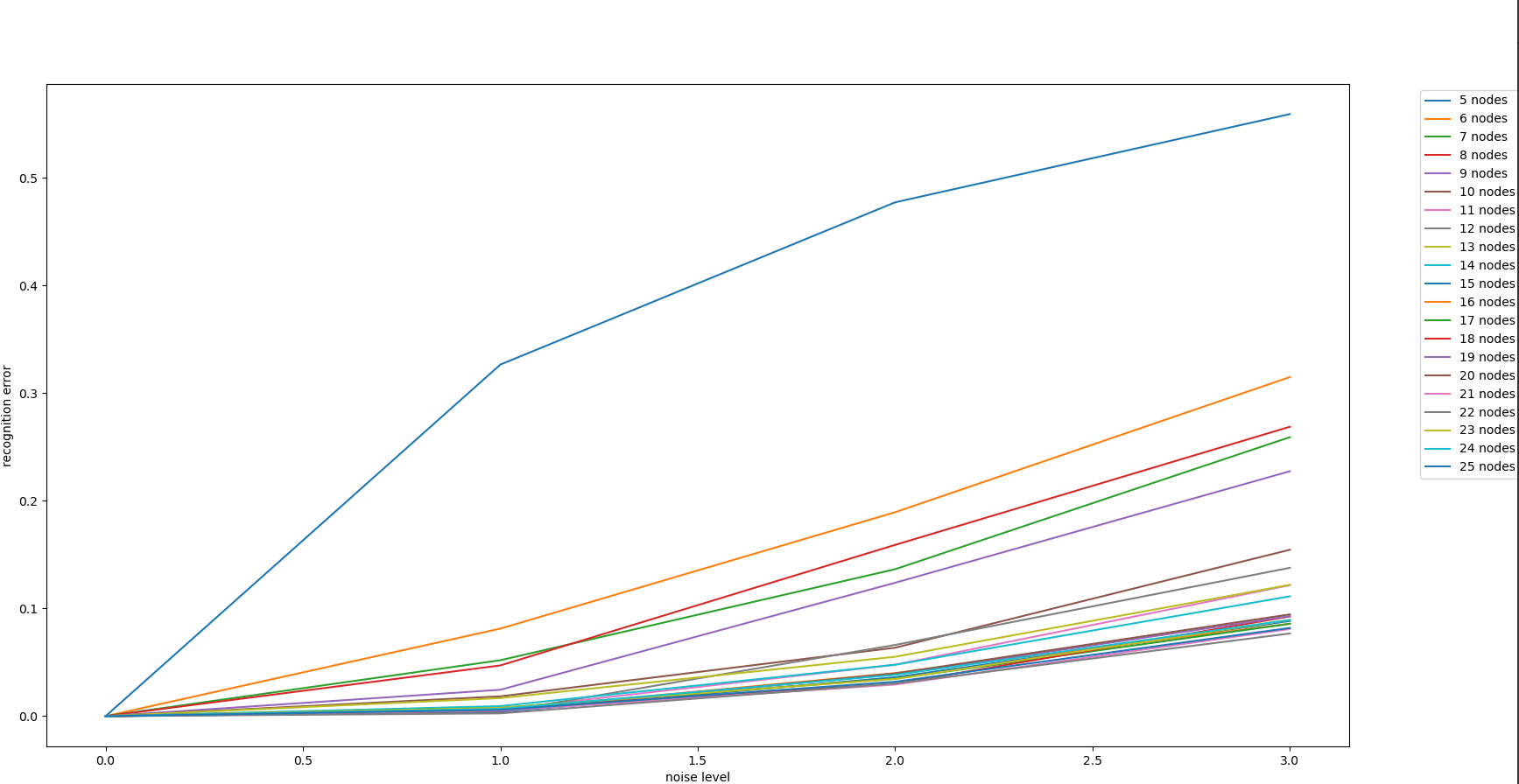


Figure 7 - Evolution of Recognition Error against Noise Level

In general, the best results for recognition rate seem to occur when our hidden layer has between 20-25 neurons. If we plot just those networks, we get a clearer picture of what our optimal size is:

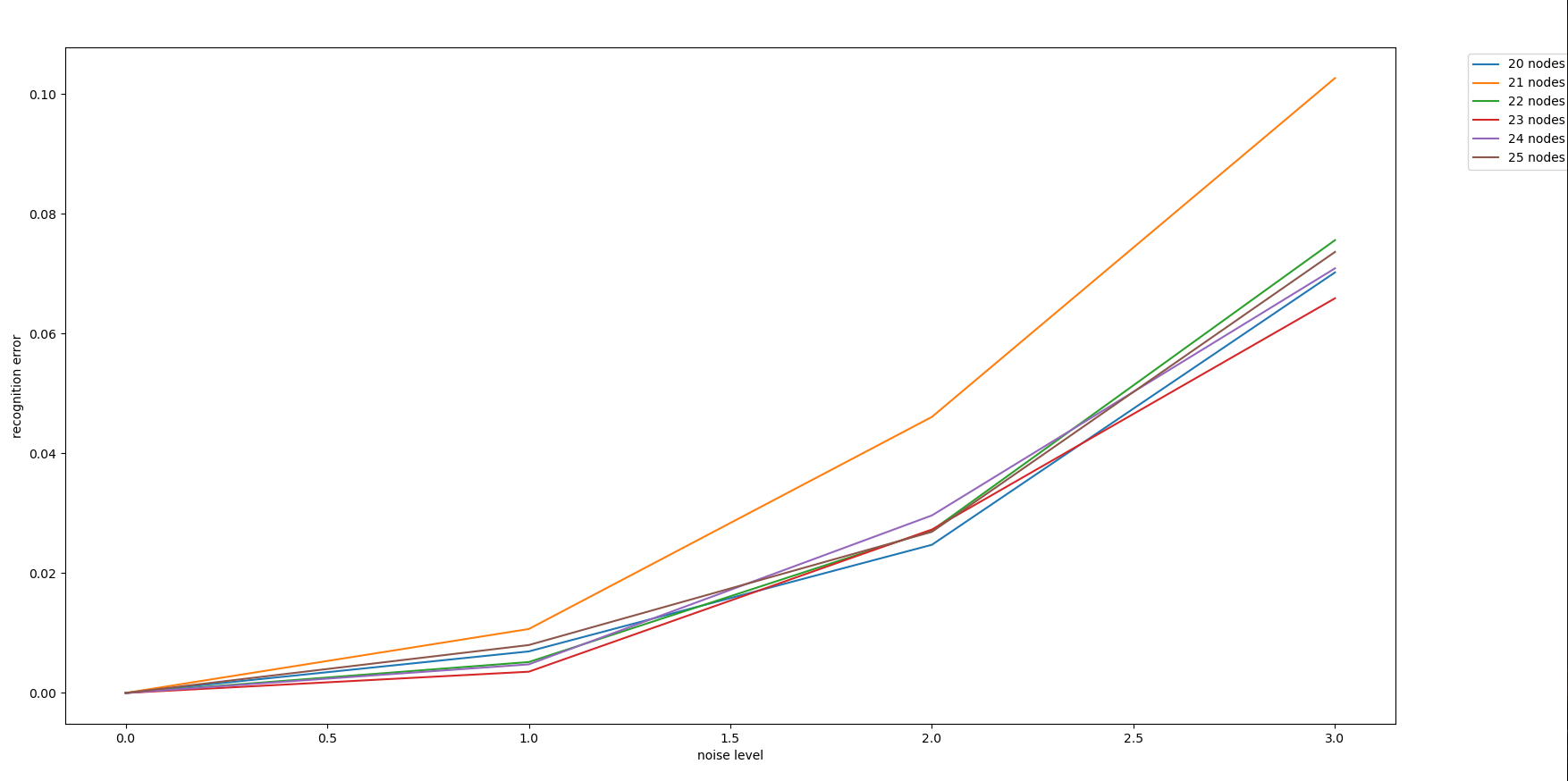


Figure 8 - Evolution of Recognition Error against Noise Level

In general, all hidden layer sizes between 20 - 25 perform relatively similarly with 23 being the best by a slim margin. These values can also vary somewhat as the noisy data is randomly generated. For the sake of simplicity, we will say that our optimal hidden layer size is 23, although it is not true 100% of the time.

## Part (b) – Network Training

1. Confirm that Fig.13 is a reasonable representation of performance for the optimal number of hidden layer neurons chosen.

Answers to 1 and 2 are both written below

1. Plot a chart in support of (1)

Figure 13.a is definitely a reasonable performance graph for our 23-neuron network. The following is an image of a 23 hidden neuron network trained on ideal data for a similar amount of time as the network in fig 13:

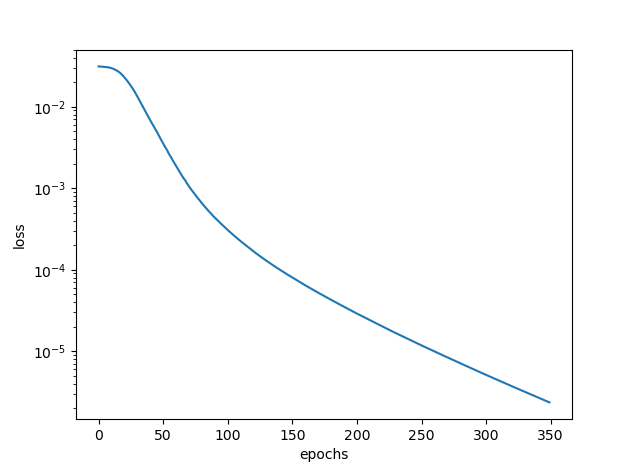
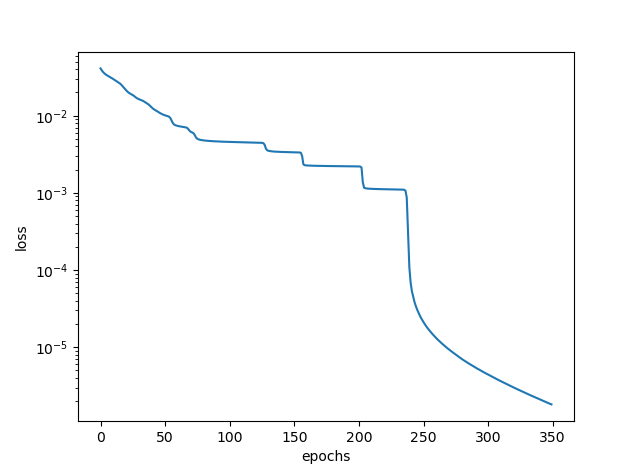
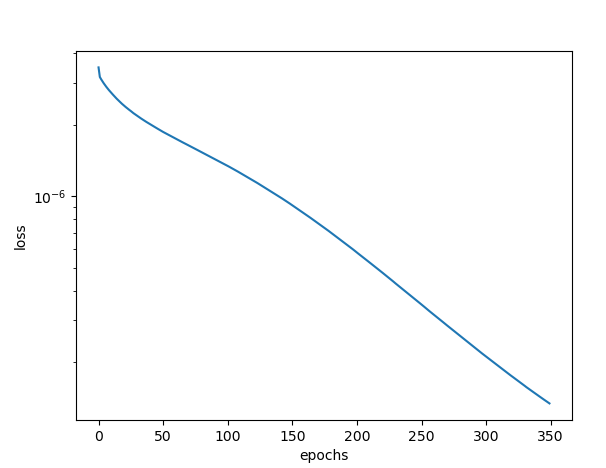


Figure 9 - Evolution of Loss against Epochs for 23 Neuron Network

While the graphs are not exactly the same, the general rate of improvement is. It should be noted that this graph can vary depending on how our weights get initialized. When we get an “unluckier” starting network we can see the sort of step pattern that occurs in fig 13:



As for fig 13.b, the performance seems more or less correct for a 23 hidden neuron network. The following is the performance graph of the network during training on ideal data for the second time:



Once again, this graph takes on a slightly more linear shape than fig 13.b, but I believe this can also be accounted for by the randomness that occurs when adding noise to our data.

## Part (c) – Network Testing

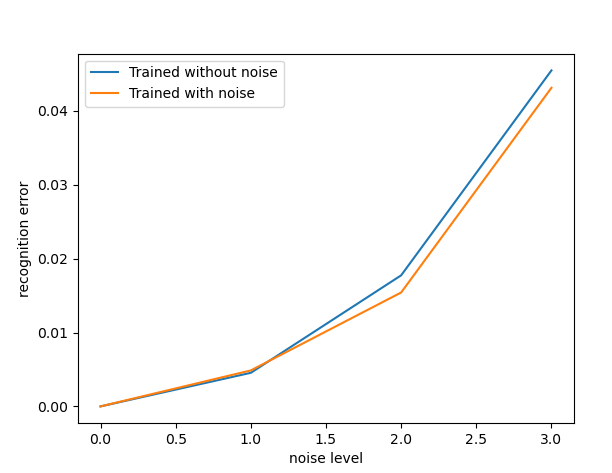
1. Create testing data that has between 0- and 3-bits noise and include some examples in your submission.

Within the submission are 4 text files called noNoise.txt, noisyData1.txt, noisyData2.txt, and noisyData3.txt.

These each contain all 31-character arrays with the relevant level of noise added, as well as the target outputs for this data. The data was generated using the same noise adding algorithm that was used for testing and training the network.

1. Confirm that you can produce the recognition accuracy shown in Fig. 14.

The following is the recognition error of a 23 hidden neuron network for noise levels 0 - 3:



The graph looks quite similar to fig 14, with the main difference being the actual recognition error values. I believe these are lower than fig 14 due to a difference in size of the training set, however the general shape of the graph is the same.