CSC3423 Biologically Inspired Computing

Callum Simpson

B6030326

Description

In this report, I will be discussing how a Genetic Algorithm and Neural network machine learning engine can be optimized in order to achieve a high accuracy in a Ying-Yang figure.

Genetic Algorithm.

A Genetic algorithm is an algorithm inspired by natural selection and genetics. It works by refining a pool of solutions to gets its most optimal fitness. It works by performing an initial step then 4 repeatable phases onto a population in a cycle.

Stage 1 is initial population. In this step a random population is created. A population are possible solutions to a problem, normally represented as a vector of elements called a chromosome. A chromosome is made up by individuals called genes.

The following stages are repeatable in a cycle based on the number of iterations the genetic algorithm is chose to run for.

Stage 2 is Evaluation (Fitness function). Each individual in the population is evaluated to see how fit it is. These fitness scores will be used to determine the probability that the individual will be selected for reproduction.

Stage 3 is Selection. This represents natural selection where the best (most fit) individual will survive and be passed onto the next generation and the weak individual will be removed. There are many different types of selection methods.

For example, Roulette Wheel Selection – A "wheel" is contrasted made up of everyone whose %size is made up of their fitness values. Meaning those with larger segment sizes will have a higher chance of being selected (fitness-proportionate selection). There is also torment section where individual will complete with one another with the winner being the one with the best fitness (rank-based method).

Stage 4 is Crossover. In this phase 2 parent individual are chosen from the population and based on a certain probability will create two offspring. There are many different types of crossover.

1 – point crossover (or single point crossover)-- Select a point in with the chromosome and swap the genes on either side of that crossover point with the other parent to create two children.

2-point crossover – Similar to 1 – point but take 2 points instead.

Uniform crossover – decide gene by gene which parent we get it from.

Stage 5 is Mutation. In this stage we make changes to a gene in a chromosome by flipping it (changing it from a zero to a one). This is done based on a low random probability (probability mutation). This done to provide some diversity within the population.

After the step, the population will be passed to step 2 to start a new cycle. [10] [11]

Lecture notes

Justification

I have chosen this Algorithm for several reasons. Firstly it was something that I had optimized in the first coursework meaning that I already have a good idea on how I could optimize it, also I still have the code and methods on how I did it in the first CW so I can simply reuse said code. This will save time which is beneficial to me as there are other deadlines that I am trying to keep a top of. As I have already used this algorithm before I already know certain combinations of values that I can test to see if I could get a good percentage quickly (I also know bad combinations meaning that I avoid wasting time on bad paths).

However, this is totally different application to use GA in so I am curious to see if I can learn anything new from it and to see how it holds up in different environment (and when given no constraints i.e 1 million budget).

Tuning process

For this algorithm I did a parameter sweep for each parameter to find a refined optimal set of results (systematic sweeps). For each test I am running it 30 times to calculate the average of said test. Whichever value gave the best average suggest that it will be the best value for that parameter. When I discover an optimal value for that parameter, I will keep that value and move onto the next set of parameters.

As each of the parameters are individual from one another then increasing should not have an impact on any of the other values.

After I have done these sweeps I will do a big brute force parameter sweep of a combination of values to se if I can find anything interesting.

Defaults

With the given inputs I received the following values:

Numberof Interaction	Accuracy	Error Time	Time
16	96.85789	2.842105	3.4505

With the following

population Size	Tournament Size	Mutator	SPC
1000	10	0.25	0.9

What interesting with these values are they are close to the already optimized values that I came across in coursework 1. The accuracy is already high so I am predicting any modifications will only be producing the smallest change in accuracy.

Population Size

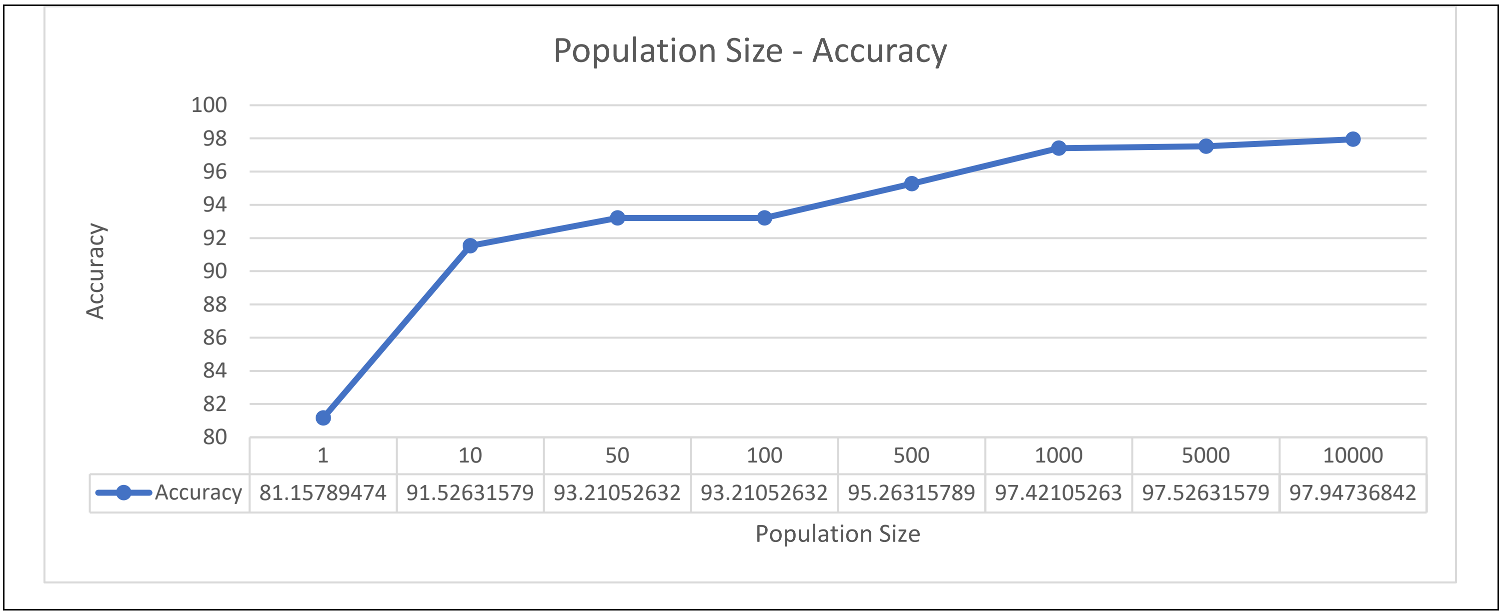
First, I tested the following Population sizes. I wanted to do this one first as I predicted that an increase in population size would lead into an increase in time. I wanted to do a sweep of some large values; these are the averages.

Population size	Iteration	Accuracy	error	Time / seconds
10	16	91.78947	8.210526	0.2386
100	13	92.31579	7.684211	0.306
1000	16	97.26316	2.736842	2.686
10000	18	98	2	34.445
100000	17	98.52632	1.473684	449.6918
1000000	18	97.89%	2.11%	8190.928

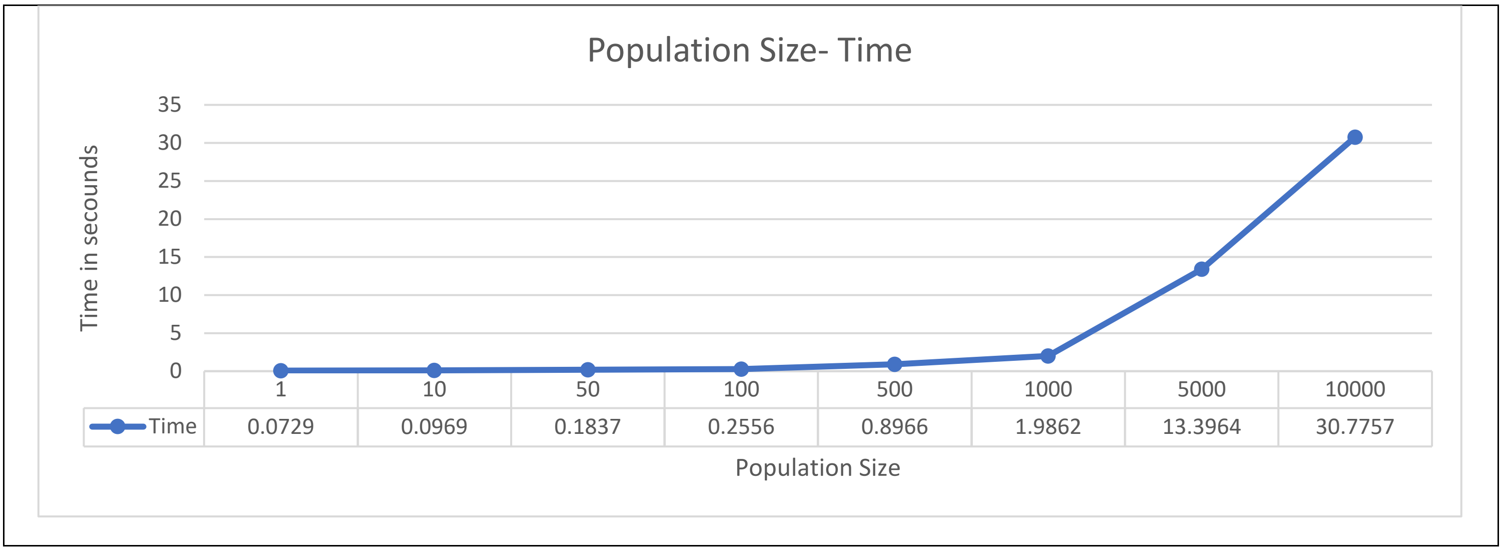
From these results, the data shows that high population size tends to yield a high accuracy. However, accuracy seems to stagnate after a point. After 1000 population the increase in accuracy seems to stagnate whilst the increase in time seems liner increase.

10,0000 seems to produce a very good accuracy, however that would take a long time to use in following parameter sweeps. I will remember this value for the end. As the values stagnated around 10000 then I will do another test

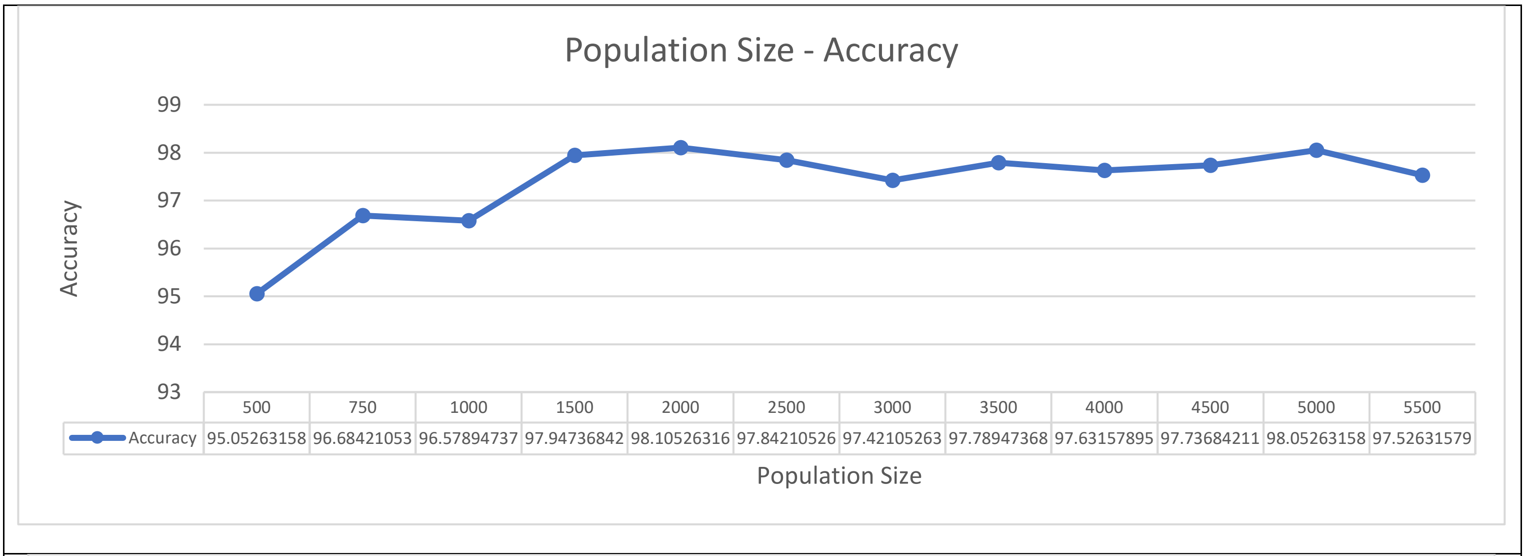
I decide to cut my search down a bit to try and find a peak with a smaller time cost.

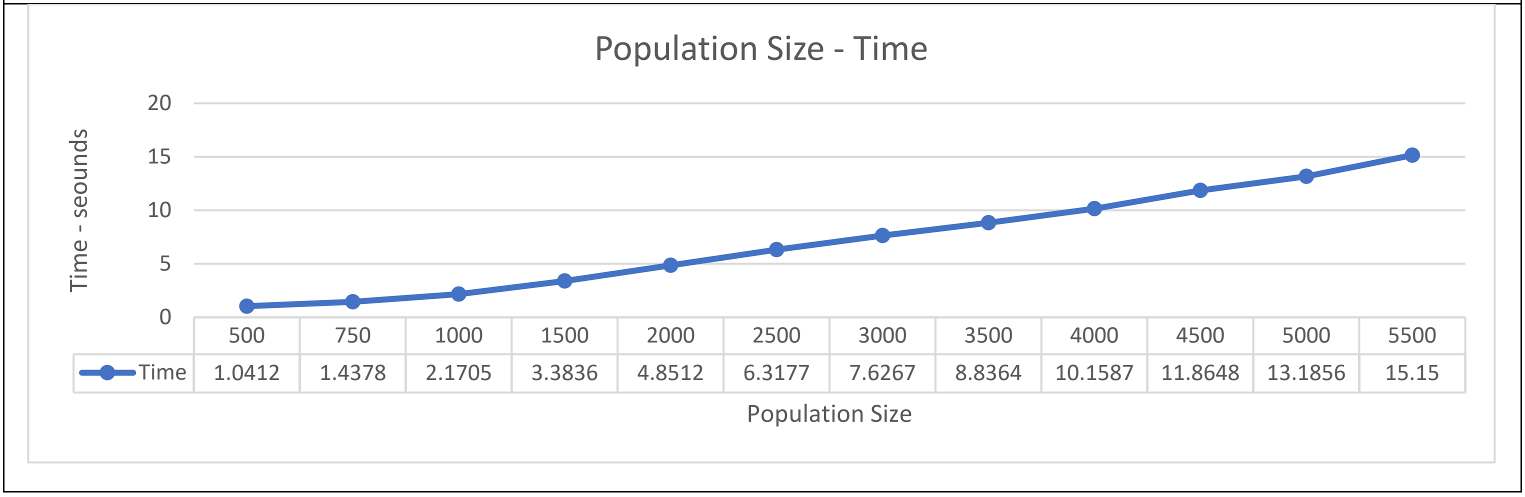


	1	10	50	100	500	1000	5000	10000
Accuracy	81.15789474	91.52631579	93.21052632	93.21052632	95.26315789	97.42105263	97.52631579	97.94736842



I ran this test a few times. All of them gave a similar result. 10000 did get the best time but as the algorithm is based on a bit of randomness, I wanted to do another sweep between. If I could find an optimal value between 500 and 5000 as that could cut down the time a lot as the data also shows that increasing the population size can have a massive impact on the time. I chose this range as the increase in accuracy stagnated after 1000 suggesting that there may be a peak around that value.





From these graphs there are two main peaks, one at 2000 and one at 5000. Out of these I will chose 2000 as it took the lowest time. Also, the fact that there are 2 relatively high peaks around 2000 further suggest that a peak exists in this range.

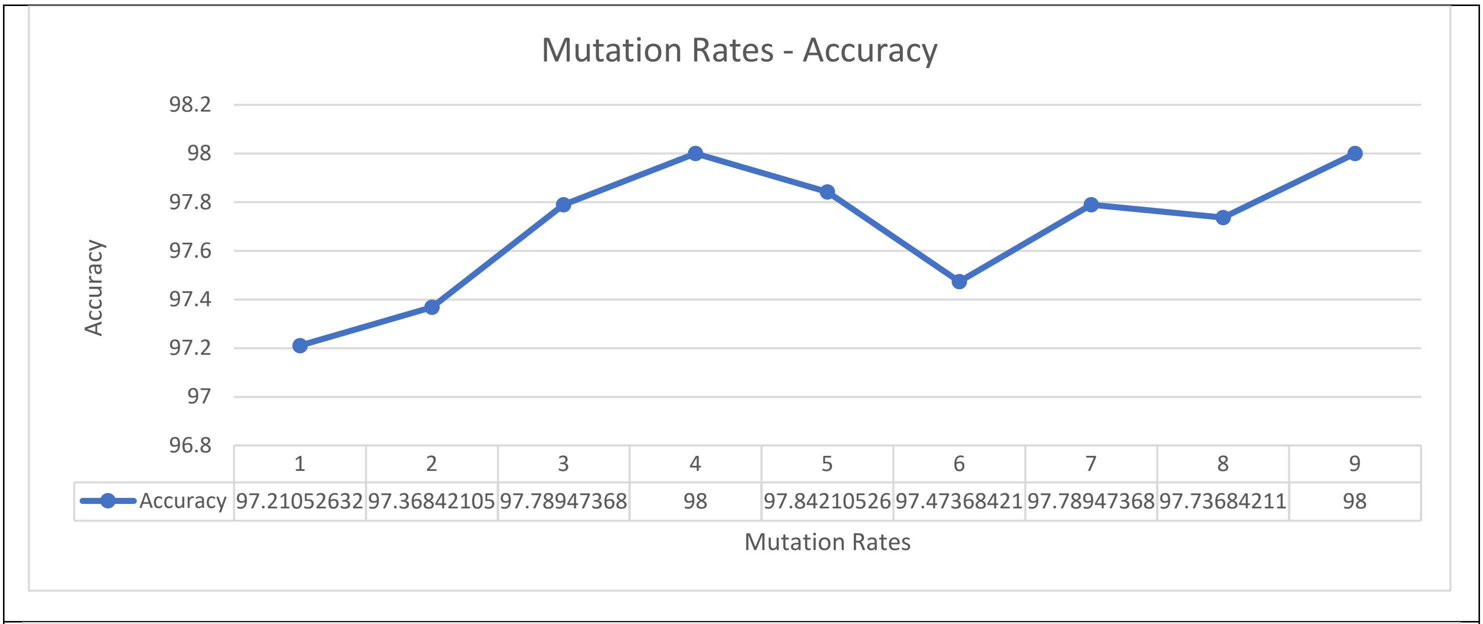
I did another sweep between 1500 and 2500 a few times and found that 2000 consistently gave the best value at 98 % accuracy.

I should also mention that increasing the population size had little to no effect on the total of number need to create the final classifier.

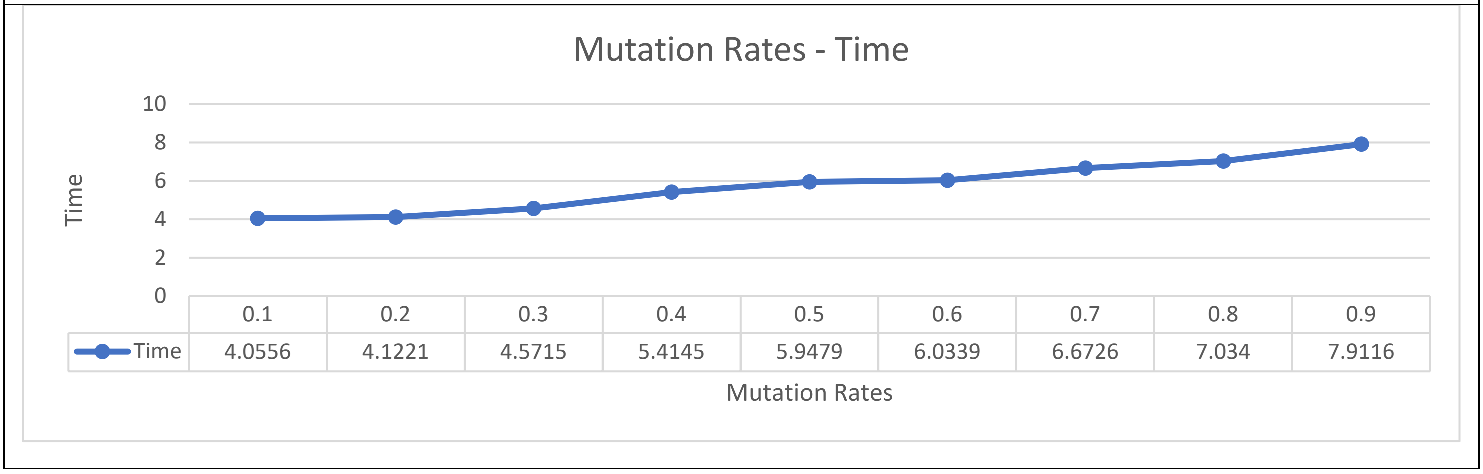
Mutation Rates

First, I wanted to do a sweep between the range of to get an idea of where the optimal value would be.

0.1 , 0.2 , 0.3 , 0.4 , 0.5 , 0.6 , 0.7 ,0.8 , 0.9



	1	2	3	4	5	6	7	8	9
● Accuracy	97.21052632	97.36842105	97.78947368	98	97.84210526	97.47368421	97.78947368	97.73684211	98



	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
● Time	4.0556	4.1221	4.5715	5.4145	5.9479	6.0339	6.6726	7.034	7.9116

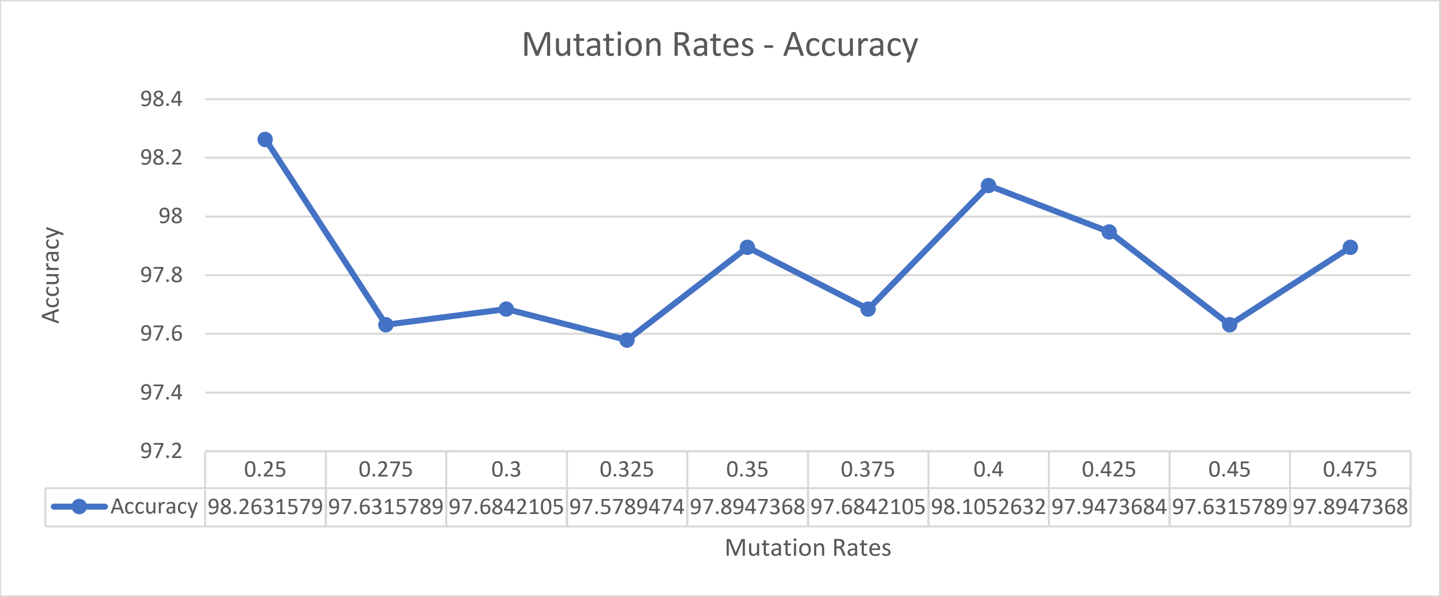
Mut R	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Ints	17	16	17	19	19	21	21	23	26

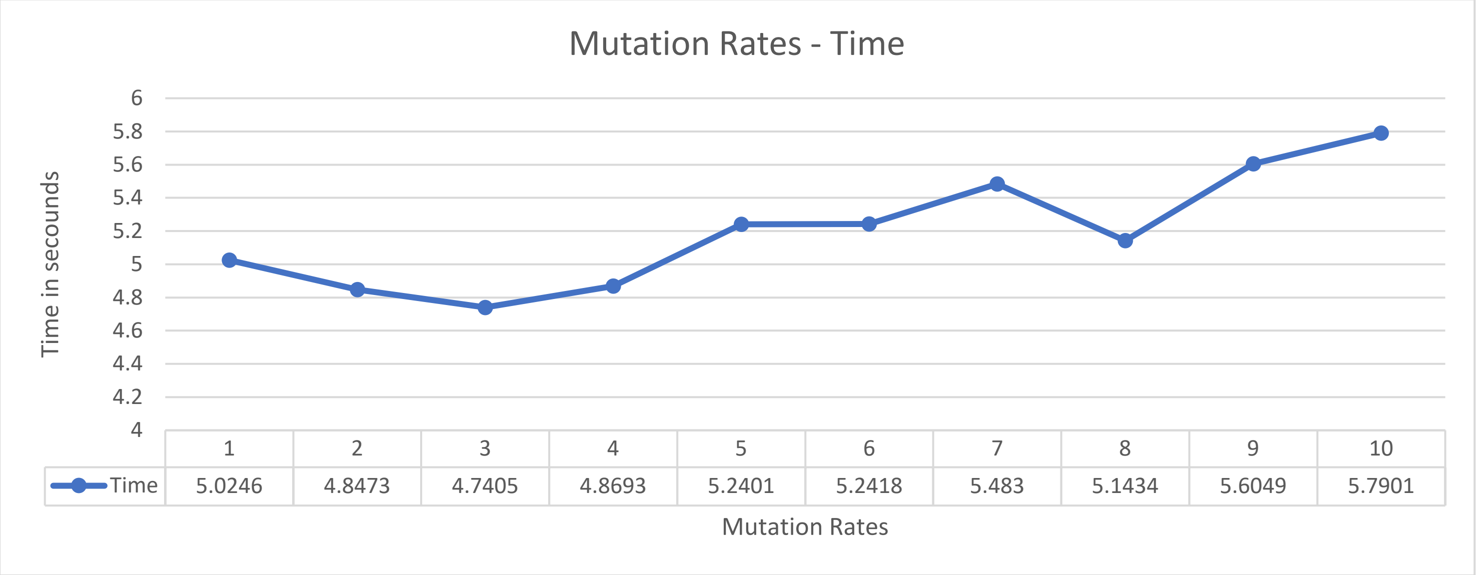
I ran this test a few times and got a similar result each time.

From the data, increasing the mutation size had a notable increase in time and a big increase in the number of iterations of the classifier. As expected, the optimal value that gets produced is around 0.4. From knowledge that I have come across, it suggests that a high mutation is potentially damaging.

I did another sweep at between 0.2 and 0.5 to see If there was a more optimal value and did the following sweep through.

0.25, 0.275, 0.3, 0.325, 0.350, 0.375, 0.4, 0.425, 0.45, 0.475





Mut	0.25	0.275	0.3	0.325	0.350	0.375	0.4	0.425	0.45	0.475
Itts	19	18	18	18	19	19	20	18	19	20

0.4 still produced a high accuracy however 0.25 produced a bit better one with a better average accuracy, lower integers, and lower time.

From these results I have deiced that the optimal value found the best value occurred around 0.25.

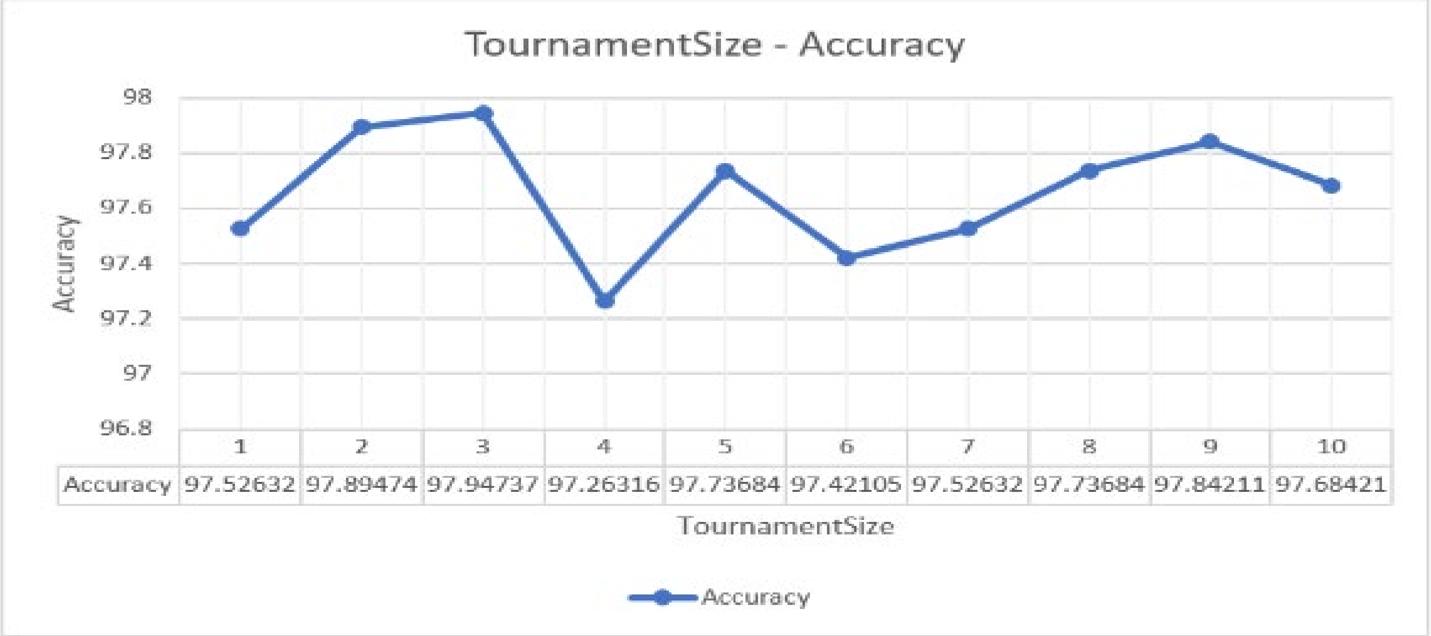
I ran a test for

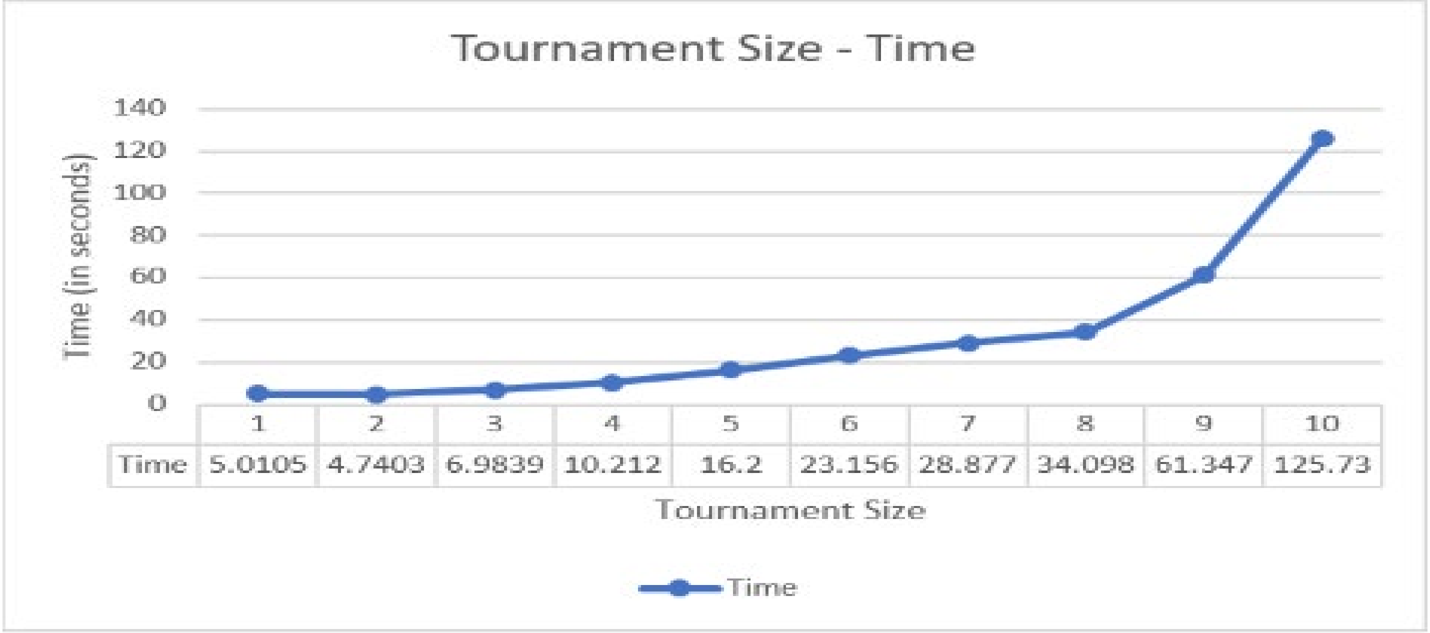
0.075, 0.1 , 0.125, 0.15, 0.175, 0.1, 0.125 ,0.15 ,0.175 ,0.2 ,0.225 , 0.25 ,0275, 0.3

The result I go from these showed that 0.25 as it best overall accuracy again. I ran this again to make sure and got similar results.

Tournament Size-

2 , 5 , 10, 50 (2.5%) , 100(5%), 200(10%), 300(15%), 400(20%) , 500(25%), 1000(50%) , 2000(100%)





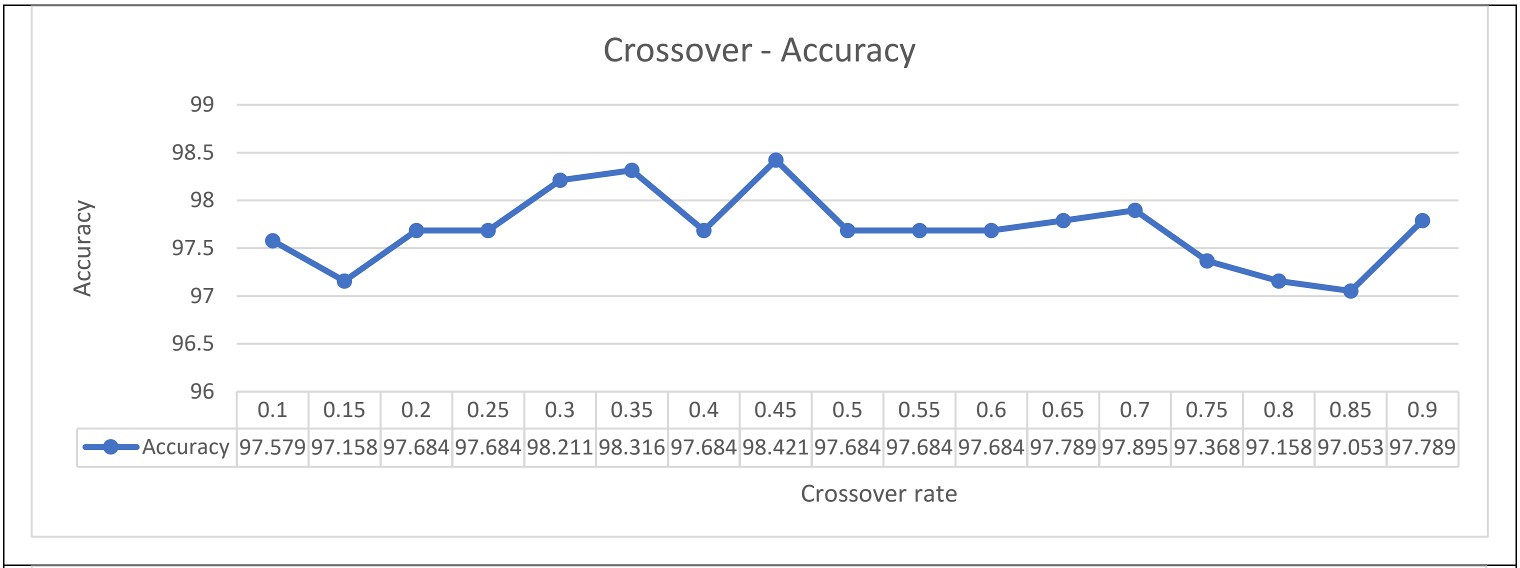
2	10	50	100	200	300	400	500	1000	2000
22	19	18	18	18	19	18	19	18	18

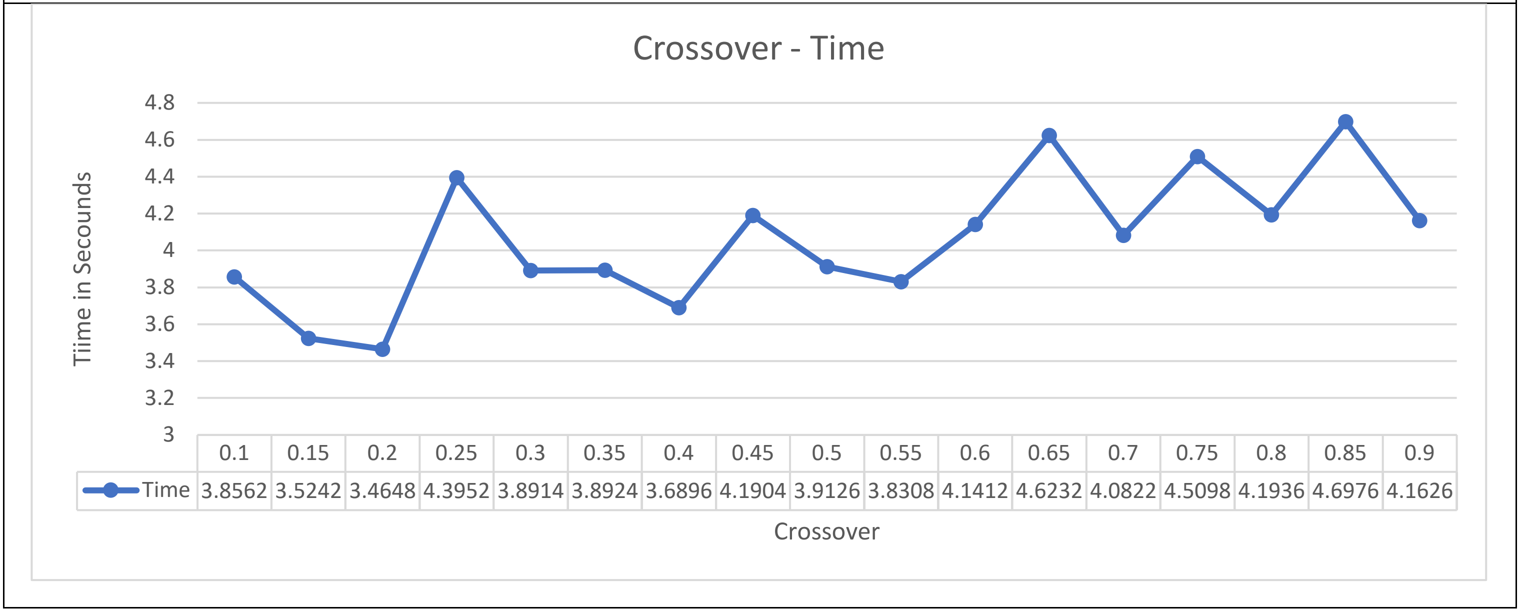
As you can see when compared to one another, Tournament size does not seem to have a huge impact the overall accuracy (the difference between the best and the worst was about 0.6%). This suggest that tournament size does not have an extremely large impact on accuracy. However, it does seem to have a large impact on time. Because of this I have chosen to keep the final tournament size of 50 (2.5% of 2000). It has a relatively low time and low iteration count.

Crossover

First, I wanted to do a sweep between the range of to get an idea of where the optimal value would be. I stayed away from 1 as from research in CW in knew 1 could potentially be harmful.

0.1, 0.15, 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 , 0.55, 0.6 , 0.65, 0.7, 0.75 ,0.8, 0.85 , 0.9





From my test I discovered that crossover did not seem to increase on the overall accuracy, 0.45 gave me 98.5% however it seems to be a sharp peak in what seems to be the start of some stagnation. I decide that I wanted to do a test to see if crossover effects mutation.

Does crossover effect mutation

In my last few tests I was looking for how the crossover and mutation individually effected the accuracy individually. However, I wanted to test is sweeping them at the same would give a better result.

Crossover 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7 ,0.8, 0.9
Mutation 0.1, 0.2, 0.3, 0.4, 0.5 , 0.6 , 0.7 ,0.8 , 0.9

Here are my top 5 results

Population	Torn	Mutator	SPC	Iterations	Accuracy	Error	Time
2000	50	0.2	0.5	18	98.73684	1.263158	5.9068
2000	50	0.6	0.9	20	98.52632	1.473684	8.584
2000	50	0.8	0.6	23	98.42105	1.578947	9.4974
2000	50	0.9	0.2	25	98.21053	1.789474	9.3618
2000	50	0.7	0.3	23	98.21053	1.789474	8.6078

The graph shows that the route of optimization that I have gone down is probably alright as I did get a similar value (mut 0.25 and SPC of 0.5). I have also kept these values as the had low impact on the iterations and low time.

Big parameter sweep

I swept through the following values all at the same time to see if I could find any other optimal values.

population	100, 500 , 1000 , 1500 , 2000 , 2500 ,3000 , 3500 ,4000 ,4500 ,5000
Torn size	2, 5% , 10%
Mutator	0.001, 0.05, 0.1, 0.2 , 0.3 ,0.4 ,0.5, 0.6 ,0.7 ,0.8,0.9
SPC	0.001, 0.05, 0.1, 0.2 , 0.3 ,0.4 ,0.5, 0.6 ,0.7 ,0.8,0.9

The top 3 results from this sweep where.

Population	Torn Size	Mutator	SPC	Iterations	Percentage	Error	Time
2000	100	0.5	0.5	18	98.72105	1.578947	6.9288
1000	100	0.7	0.5	22	98.63158	1.368421	7.1772
1000	200	0.8	0.9	24	98.63158	1.368421	11.192

Obviously, this is a bit more of a brute force way of finding the optimal value, but it was easy to set up. The sweep didn’t use as refined values as my systematic sweep, but I did find an overall optimal value that was close to what I found in my systematic sweeps. Because of this I wanted to further refine my value by doing the following.

Final iteration – I swept through the following values

population	1500, 1600 ,1700, 1800 , 1900 ,2000, 2100 , 2200 ,2300 ,2400
Torn size	2, 2.5%
Mutator	0.2, 0.225, 0.250 , 0.275 , 0.3
SPC	0.4, 0.425, 0.450 , 0.475 , 0.500

From that search this was the top 3 value are.

Population	Torn Size	Mutator	SPC	Iterations	Percentage	Error	Time
2200	55	0.275	0.45	17	98.42105	1.578947	6.9288
2300	57	0.25	0.5	19	98.42105	1.578947	8.1946
1600	40	0.275	0.475	19	98.31579	1.684211	4.7558

At this point I am barley getting any more changes, this suggest that I have reached an optimum.

Knowledge representations –

I will admit I had trouble setting this up to represent different shapes, but I believe that the one we are currently using (the sphere) is the most optimal knowledge representation. This is because we are optimizing the Ying-yang model, the curvature of the circle will fit into the curvature of the ying-yang model a lot more optimally than other shapes, square would overlap the curve or wouldn't be able to fill the curve as well. Because of this (and that I did not have time) I have decided that sphere classifier is probably the most optimal.

Performance

As this produced the best result, I have chosen this as my final value.

Population	Torn Size	Mutator	SPC	Iterations	Percentage	Error	Time
2000	100	0.5	0.5	18	98.72105	1.578947	6.9288

```
Classifier of iteration 19. Accuracy 100.00%, coverage 0.12%
Iteration 19, removed 1 instances, instances left 850
Overall stats at iteration 19. Accuracy 49.94%, error rate
0.00%, not classified 50.06%

Classifier of iteration 20. Accuracy 100.00%, coverage 0.12%
Iteration 20, removed 1 instances, instances left 849
Overall stats at iteration 20. Accuracy 50.00%, error rate
0.00%, not classified 50.00%


Classifier of iteration 21. Accuracy 100.00%, coverage 100.00%
Iteration 21, removed 849 instances, instances left 0
```

```
Overall stats at iteration 21. Accuracy 100.00%, error rate
0.00%, not classified 0.00%

Stats on test data
Accuracy 98.42%, error rate 1.58%, not classified 0.00%

Total time: 8.293 (-im not sure why there is this large increase
in time)
```

Neural network case

Description of the selected nature-inspired

A Neural Network is an algorithm modeled after the human brain to recognize patterns from a given a set of data. It is made up of multiple neurons that takes in inputs and classifies them according to the given algorithm that is applied to the NN. Normally, a NN is made with a input layer, 1 or more (though most only require one) hidden layers and an single out layer. Each layer is made up of interconnecting nodes and each node is a perception (something that takes in many inputs and produces a single output [13]) that feeds the signal into an activation function. [1]

There are many different types of NN, the one we are looking at in this test is the simple multi-layer perceptron neural network. MLPs are suitable for classification prediction problems where inputs are assigned to a class or label. [2]

MLP is a “feedforward” artificial neural network which uses backpropagation for training. Multilayer perception has been described as a “the hello world of deep learning”. Feedforward are in a constant back and forth called forward pass and backward pass. [3]

Training normally involves adjusting the parameters, or the weights and biases, of the model in order to minimize error. Backpropagation is used to make those weigh and bias adjustments

Forwardpass is when a input layer is takes in a signal and the output layer makes a prediction about the input. Backwordpass is when partial derivatives of the error function with respect to the various weights / biases are backpropagated through the MLP. [4]

The parameters of NN are

Input layer – initial data for the network

Learning rate – quickly the network updates parameters. Low learning rates a re slow but converges a lot better than a high learning rate.

Hidden layers – The number of layers between the input and output layer.

Neurons – the number of neurons that are in a hidden layer

Output layers – output results. [12]

Justification

There are quite a few reasons why I decided to use a Neural network. Firstly, it is something that I have never done, and I found the concept behind it really fascinating so I wanted to learn about it. I felt that the task of optimizing this algorithm would be straightforward and I quite quickly worked out how I would tackle it. I was pushed for time and for my quick test on NN ran very quickly so getting results would not take too long. Another reason was that I may be looking into using a neural network in my dissertation so I wanted to gain an understanding of how it worked, learning about it now will help me in the future and save me time.

Turning processes

My plan is to first find the optimal activation function as that will greatly affect how the NN operates. Once I have found the best activation function, I will then sweep through each parameter to find the best magnitude for the value and then search for the best optimal for that value.

I will do this so that I do not have to waste time looking "down paths" that would give me poor results.

However, I do think by going down this path that I will miss possible routes of optimization. I will do a large parameter sweep (though not majorly precise) at the end to see if I can discover any other path of optimization or to see if it will back up my method.

I am optimizing based on accuracy rather than time. However, If I come across options that have a similar accuracy level then I will pick the one which take the less time.

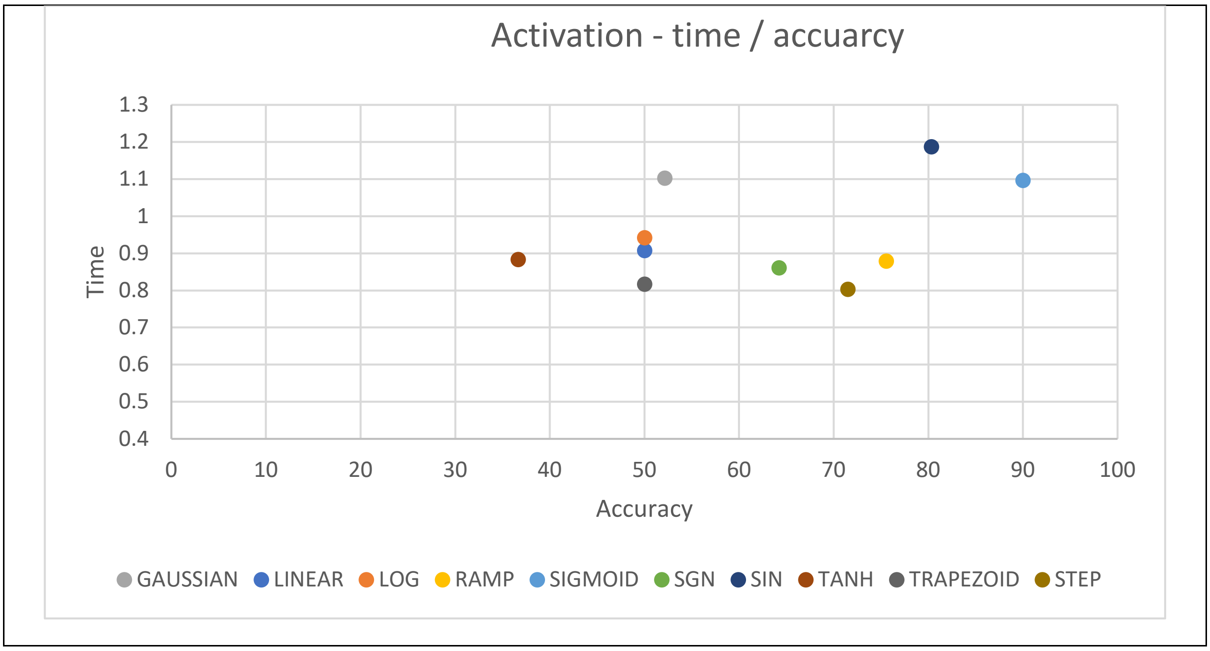
Originally, I was testing all the activation layers at the same time, however after a chat to a demonstrator I was told that I should only be picking one function and optimising after that.

*Note about my results – for each test I have run everything 30 time to calculate an average, this is to smooth out any outliers (which is import for NN as there is a little bit of randomness).

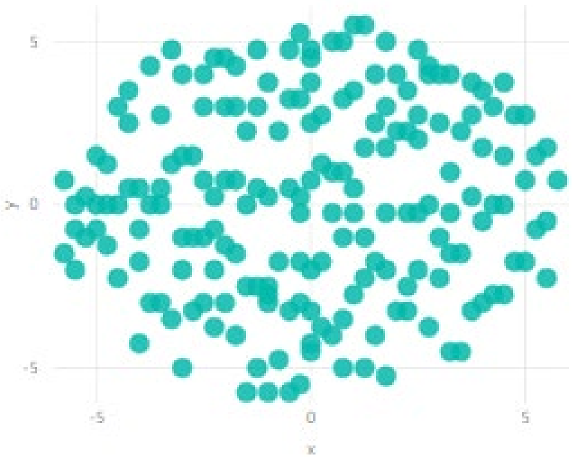
Performance – Activation function

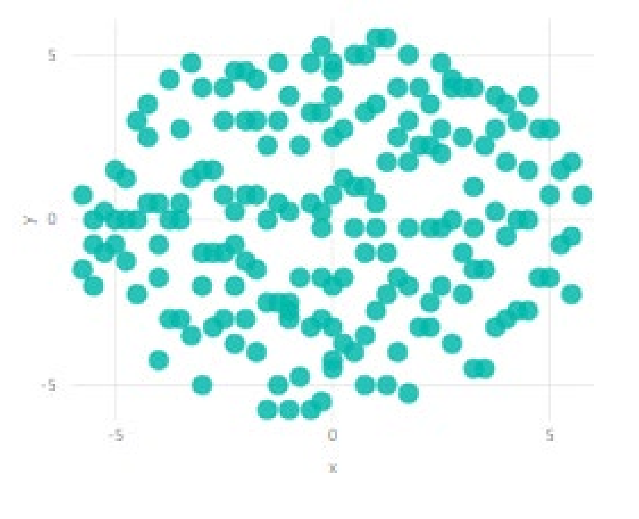
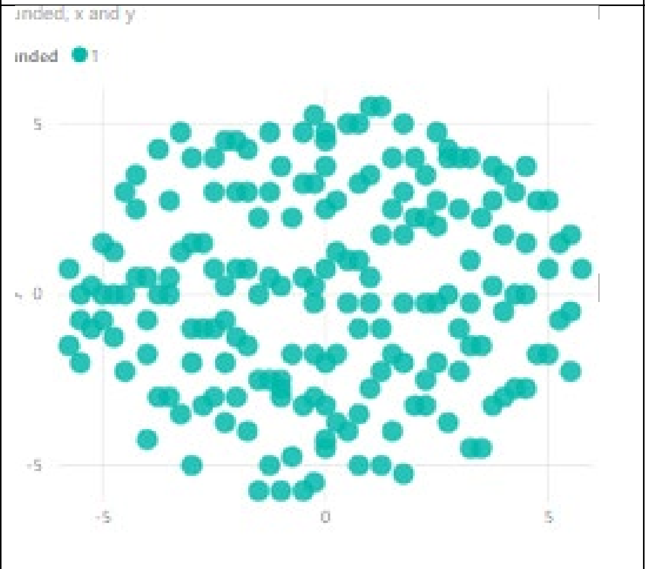
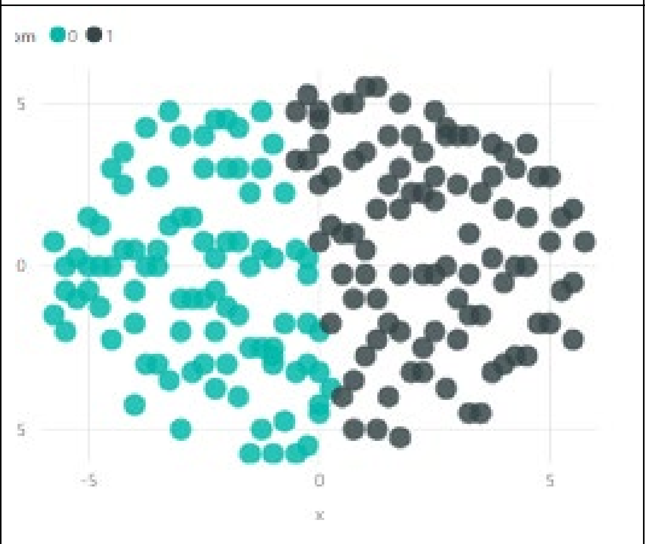
An activation function is a simple mapping of summed weighted input to the output of the neuron. It is called an activation function because it governs the threshold at which the neuron is activated and strength of the output signal. [5]

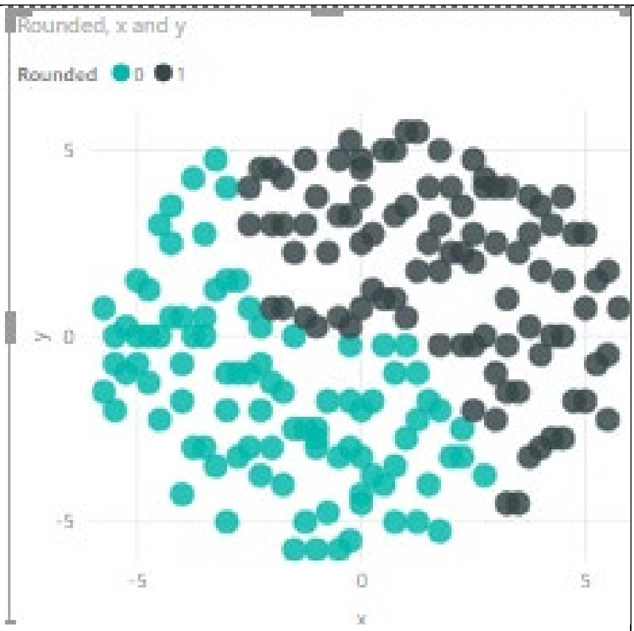
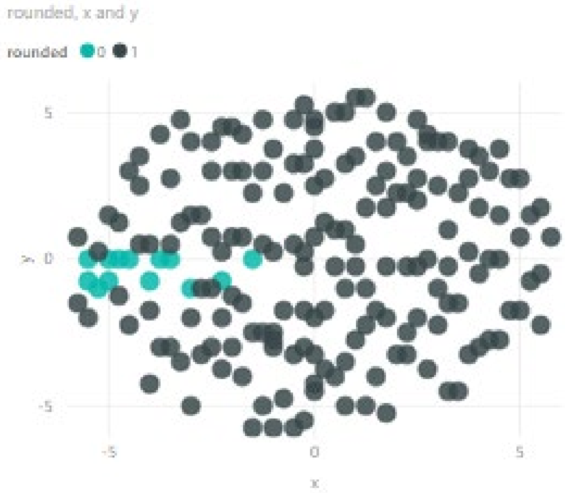
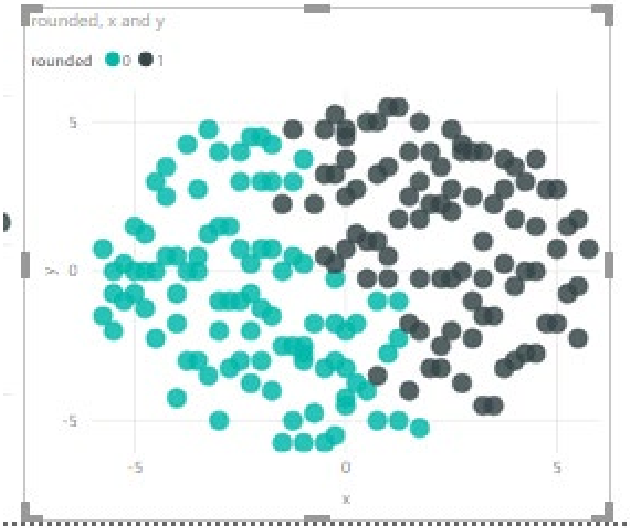
First, I wanted to test each of the actions against the given defaults to see what kind of results that they would produce (in terms of accuracy and time). The

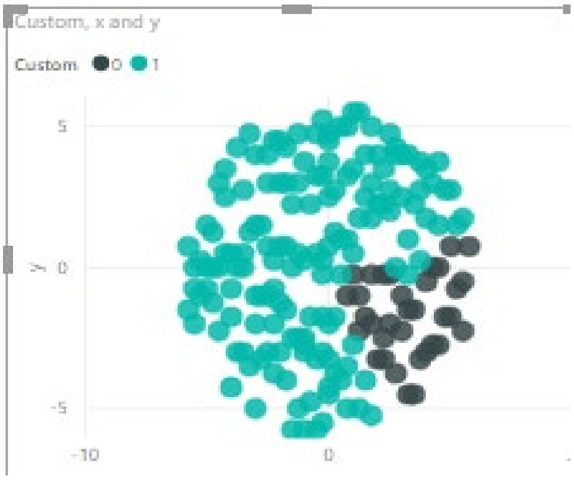
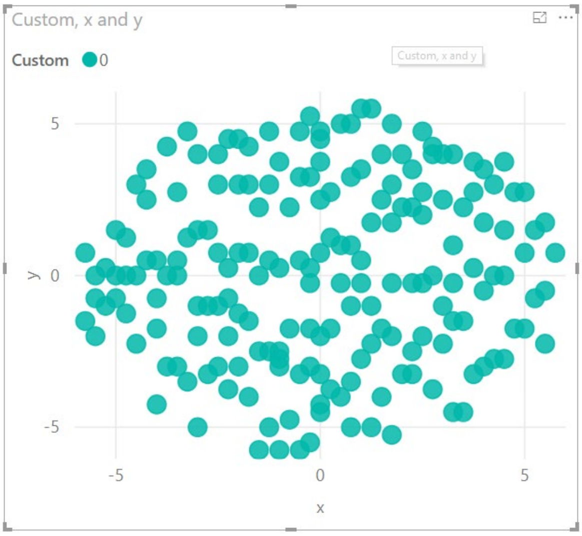


Here are the resulting Ying-Yang Diagrams that I received from running the different activation functions with the defaults (on the test data).

Activation	Pic of test data	Reason
Gaussian		Gaussian is an even function so it will give equally positive and negative values of the input. This is only 50% as it only predicted one type of color.

Linear		Linear is a straight line and It’s also not possible to use backpropagation which is what a MLP needs to do in order to learn.
Log		I couldn’t find anything for log
Ramp		ramp function is a unary real function. It allows for easy backpropagate of the errors which is good for MLP. Ramp works by not activing all the neurons at the same time. And if an input is negative it will be converted to zero. [6]

Sigmoid		Sigmoid would be very suitable as it is a curve between 0 and 1. This gave a result of 89% and you can see the ying-yang shape is already taking place.
Sgn		Couldn't find anything for Sgn
Sin		In a neural network Sin curves between 1 and – 1. [7]

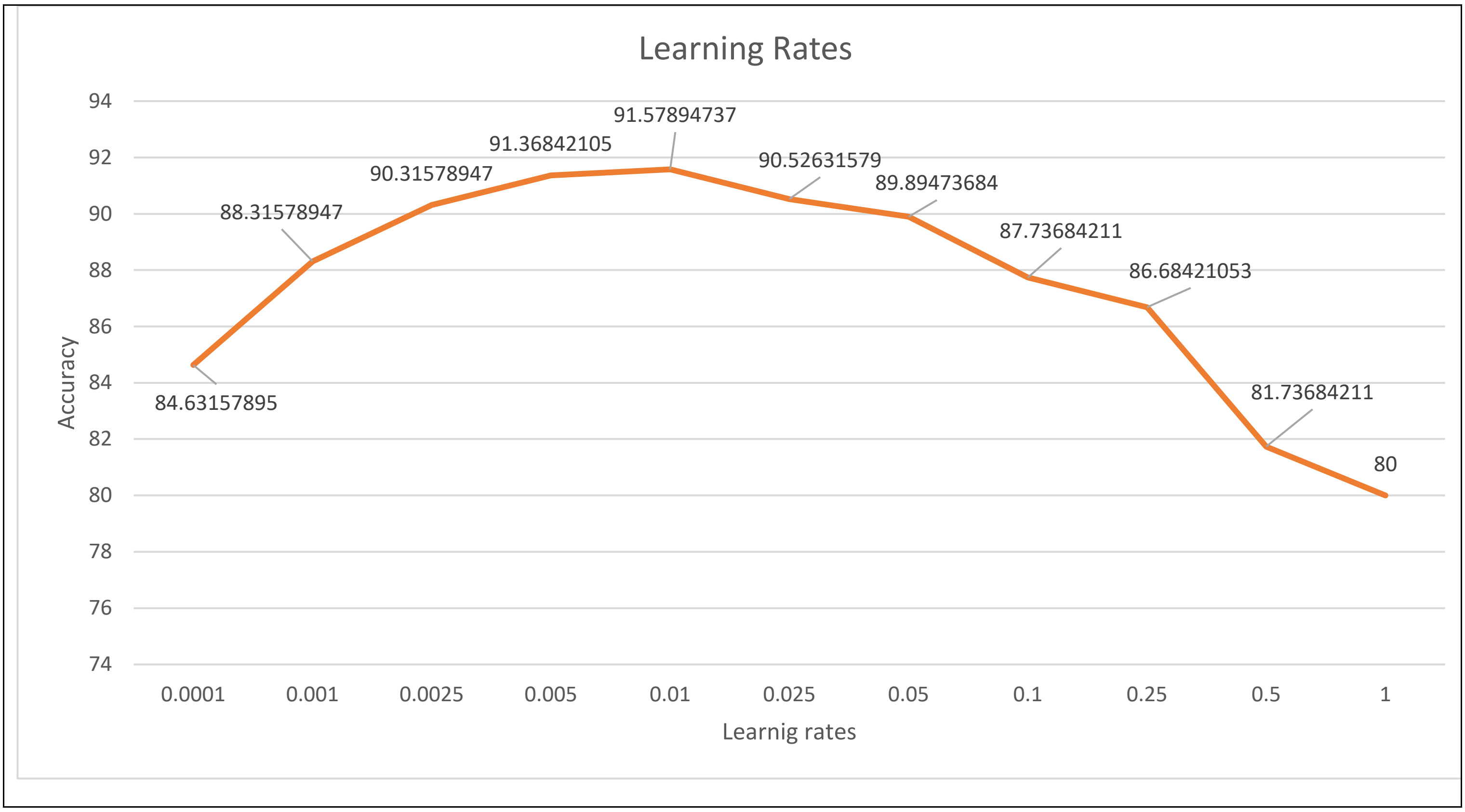
Tanh		Tanh would not be suitable as it goes from -1 to 1 so there are 3 possible values that it would return (and I only want the activation function to return 2 values)
Trapezoid		

Sigmoid arguably produced the best result out of all the actions, with the highest accuracy and a lower time than the second highest accuracy. Sigmoid would be very suitable as it's a smooth curve between 0 and 1. Also The two historically common activation functions are both sigmoid. So, it makes sense to use it. I am only going to continue using the one activation function as in previous go at this course work (one which I was testing every activation function at the same) different Activation functions react differently to each parameters, making it difficult to optimize as I go. [8]

Performance- Learning Rates

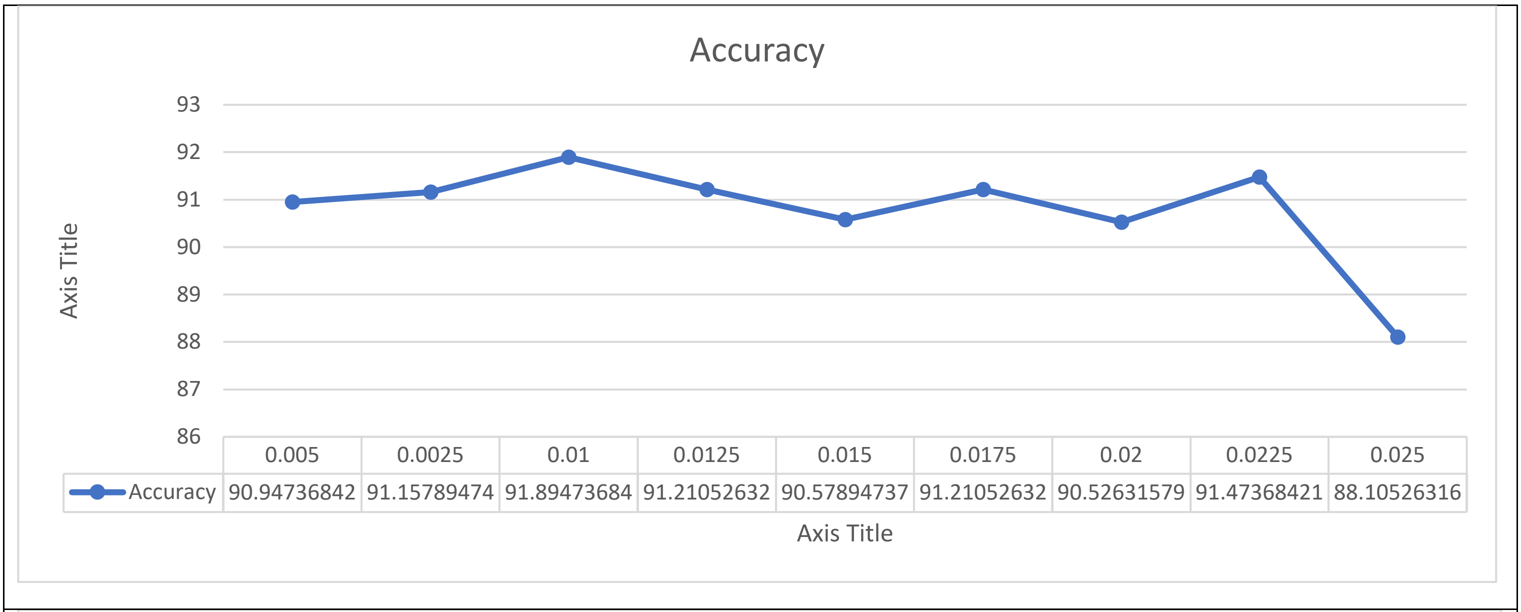
From reading an online source I discovered that learning rates should between 0 and 1. I tested this by the following learning rates over all the actions. [9]

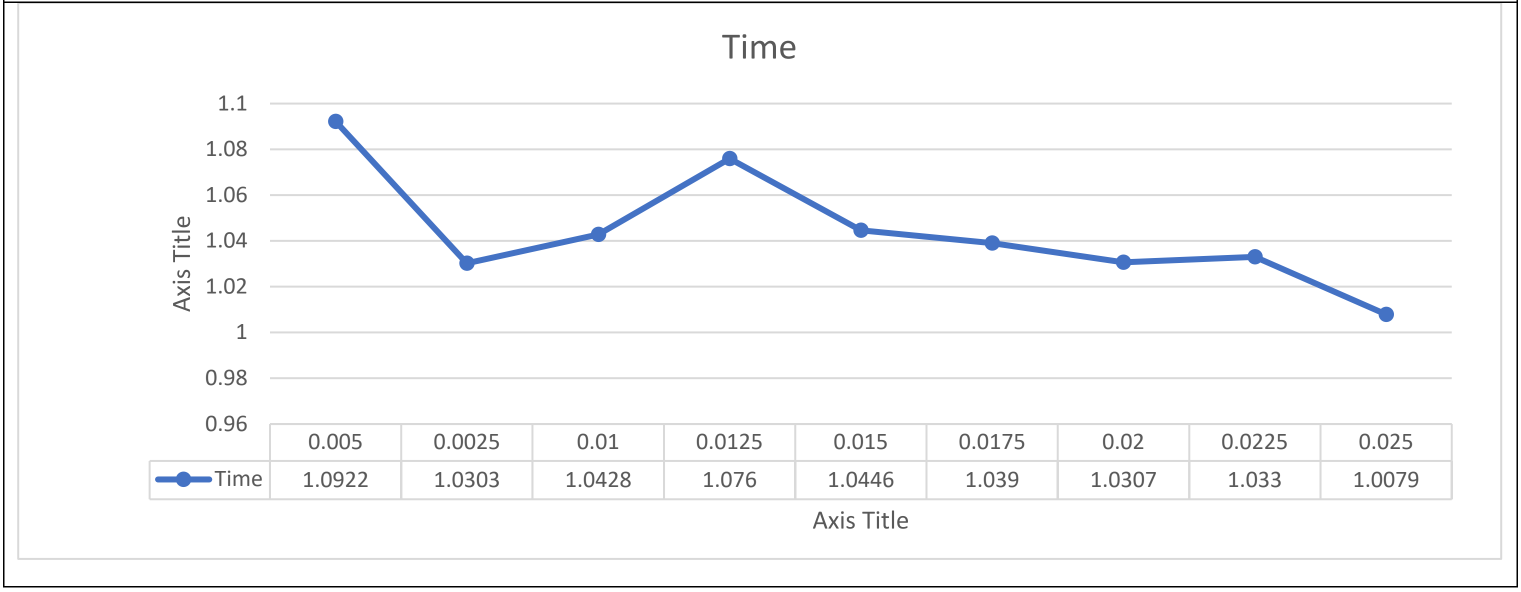
1.00E-04, 0.001 , 0.0025 , 0.005 , 0.01, 0.025, 0.05 , 0.1 , 0.25, 0.5 , 1



From my initial run it seems the best value is between 0.005 and 0.025. This led me to test the results below. It should also be mentioned that a higher learning rate resulted in less time taken to run the test.

0.005 , 0.0025, 0.001 , 0.0125, 0.0150, 0.0175 , 0.0200, 0.0225 , 0.0250





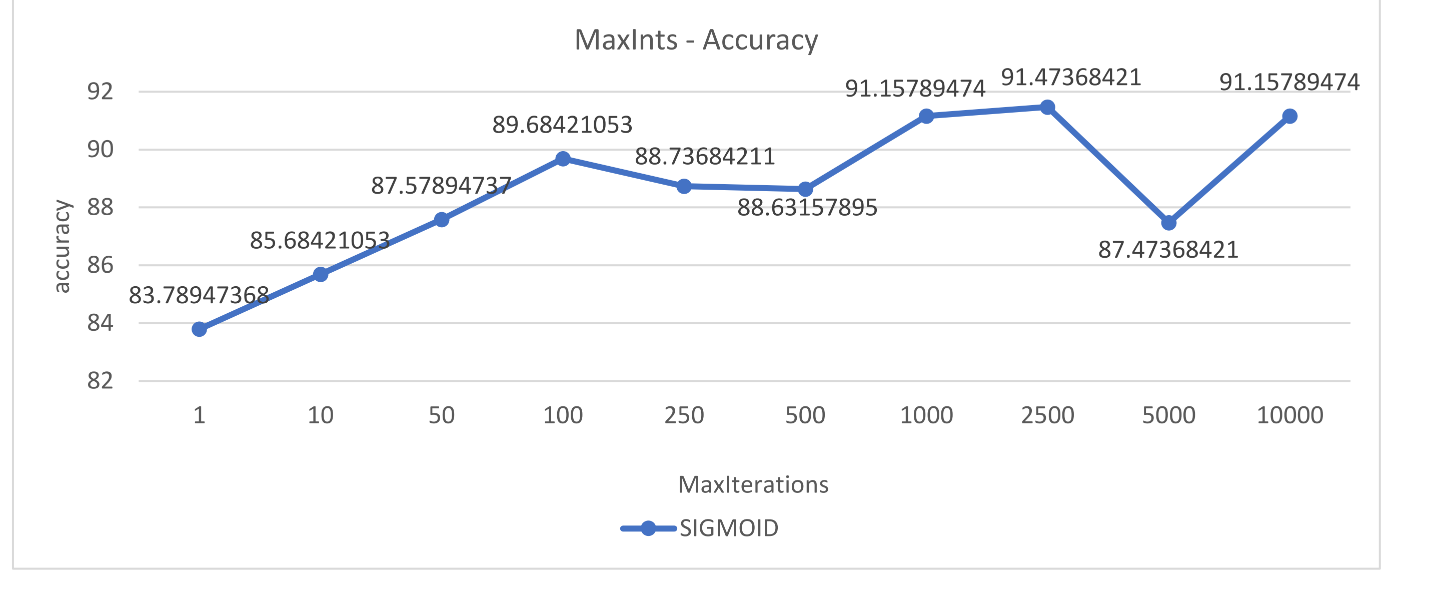
After refining my results, I achieved the following values. From looking at my results it seems between these values accuracy is only within 1% or 2% of each other. As it taken from an average of 30 I have decided to keep 0.01 as my learning rate as it took the least amount of time and got an accuracy near 92%.

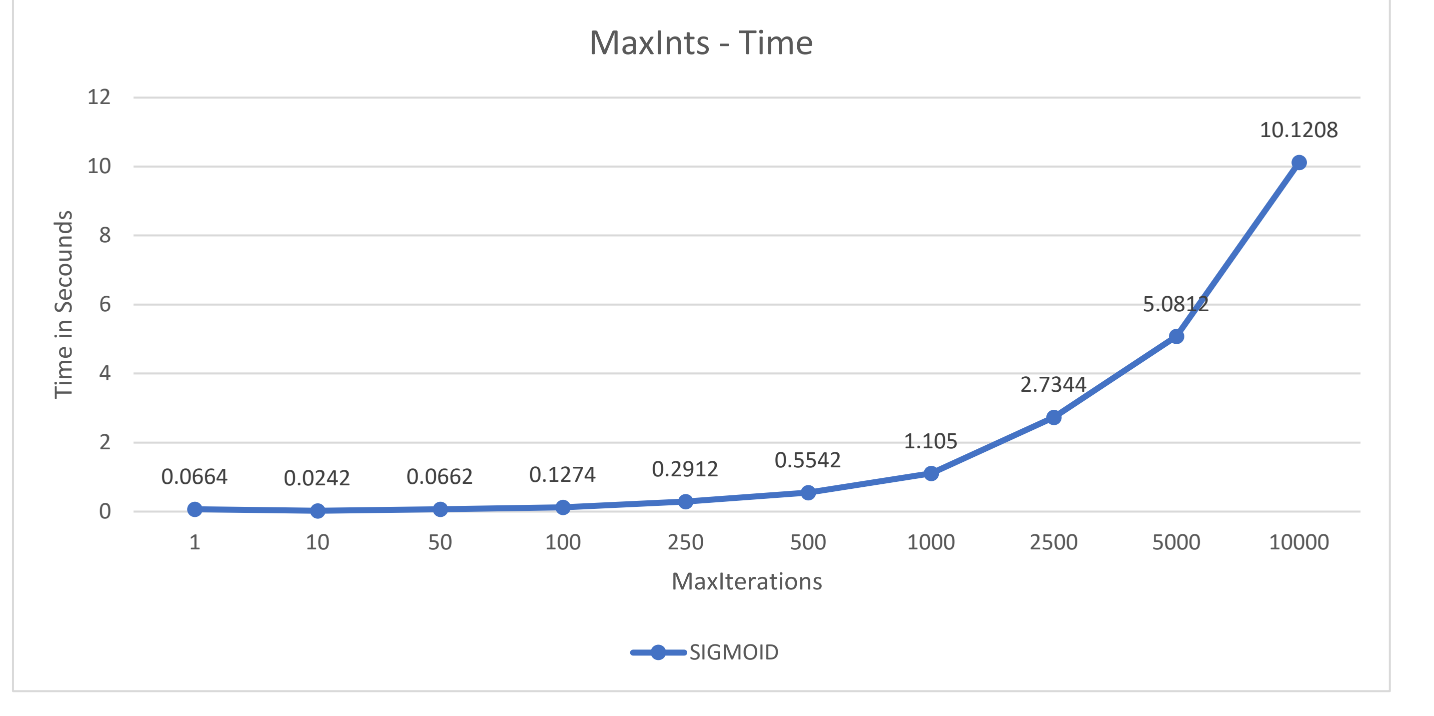
<u>Performance – output layer</u>

I read online that output layer should be one. I wanted to test this out by trying out other output layers. Testing anything higher than one caused errors so I will keep output layer at one. [9]

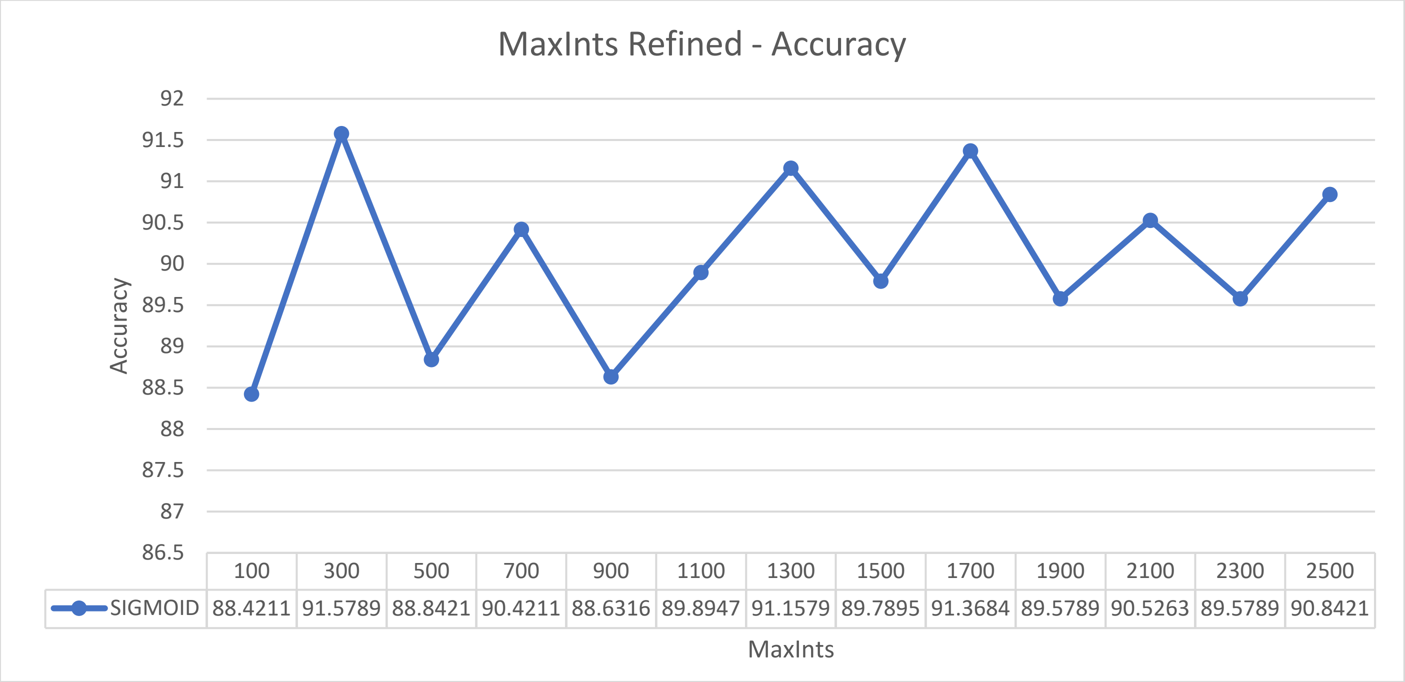
Performance – Max iterations

1, 10 , 100 ,500, 1000, 5000 , 10000

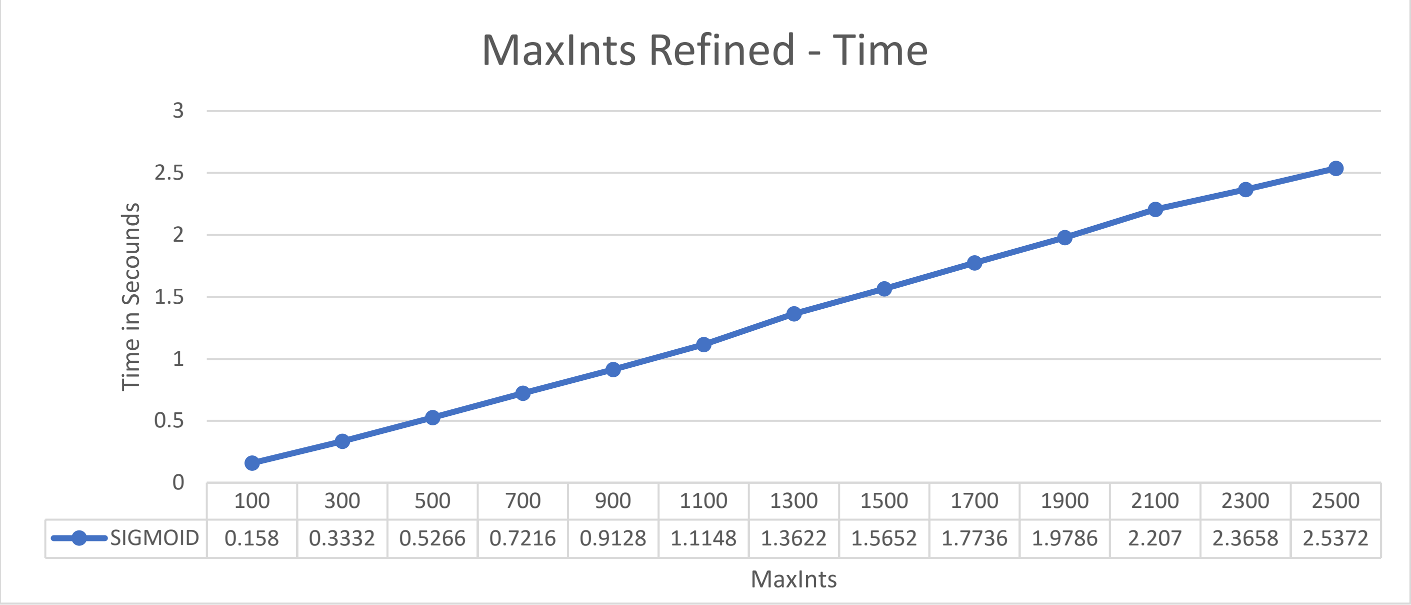




From the tests its clear to see that increasing the max integers has a linear increase in time whilst having a little effect on overall accuracy. As there was a notable peak at 100 and a notable drop off at 2500, I wanted to run another test to check between these two points to see what happened. I did not want to go over 2500 as it looks like overfitting is taken place, however I tested up to it to make sure.



	100	300	500	700	900	1100	1300	1500	1700	1900	2100	2300	2500
SIGMOID	88.4211	91.5789	88.8421	90.4211	88.6316	89.8947	91.1579	89.7895	91.3684	89.5789	90.5263	89.5789	90.8421



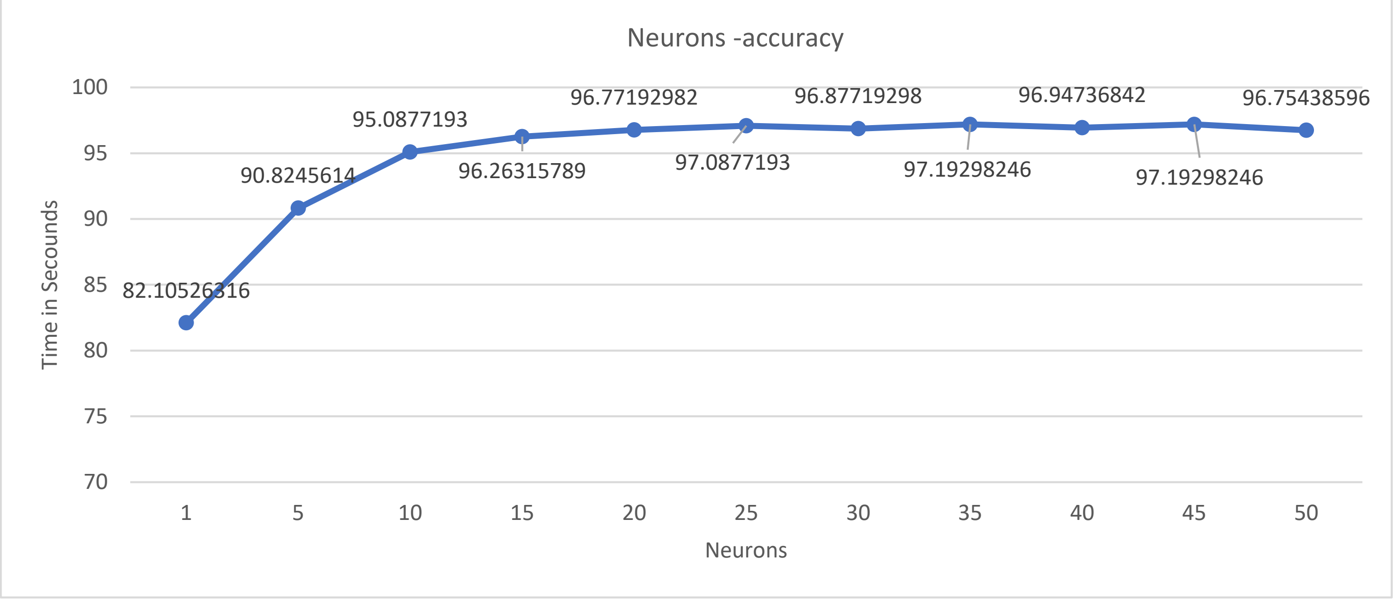
	100	300	500	700	900	1100	1300	1500	1700	1900	2100	2300	2500
SIGMOID	0.158	0.3332	0.5266	0.7216	0.9128	1.1148	1.3622	1.5652	1.7736	1.9786	2.207	2.3658	2.5372

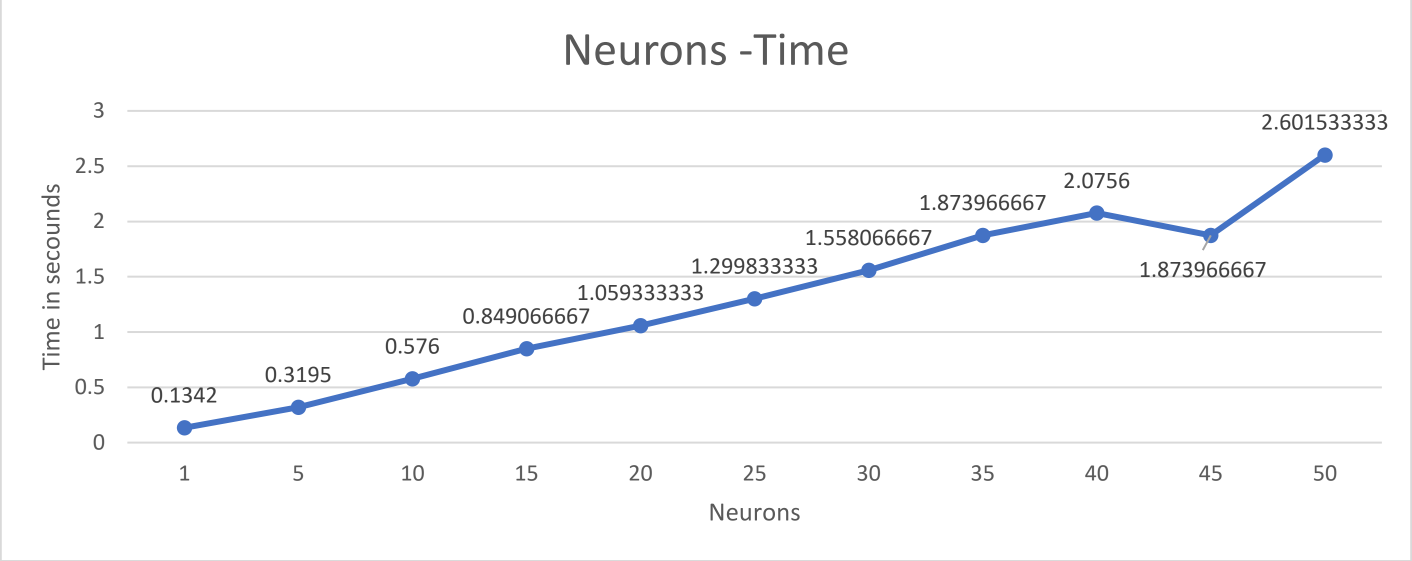
After this test I decide that 300 would be my optimal value. Though there were other values that gave averages of accuracy similar to 300 (i.e 2500) , 300 had a lot lower time to run (0.5 seconds vs 2.7 seconds).

Performance – Hidden layer neurons

I tested this by sweeping through the actions and testing the following neurons. I read online that the recommend rule of thumb method for neurons is “between the size of the input and size of the output layers”. I wanted to test a range of values to see what results that I would get. [9]

1, 5, 10, 15, 20, 25, 30 , 35, 40 , 45 ,50

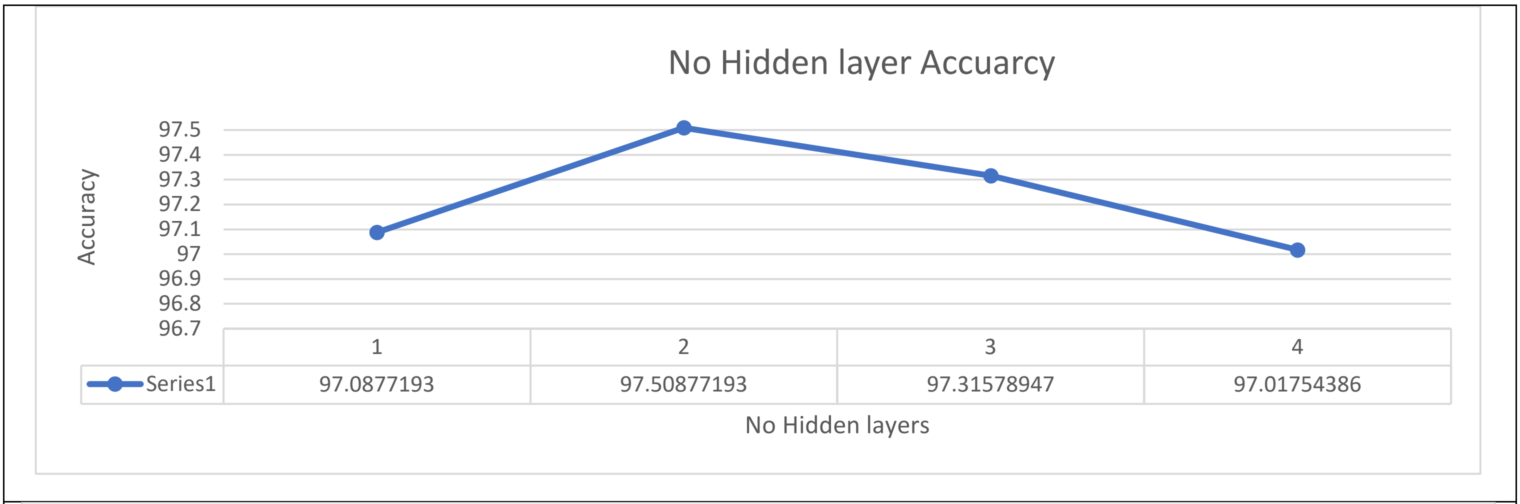


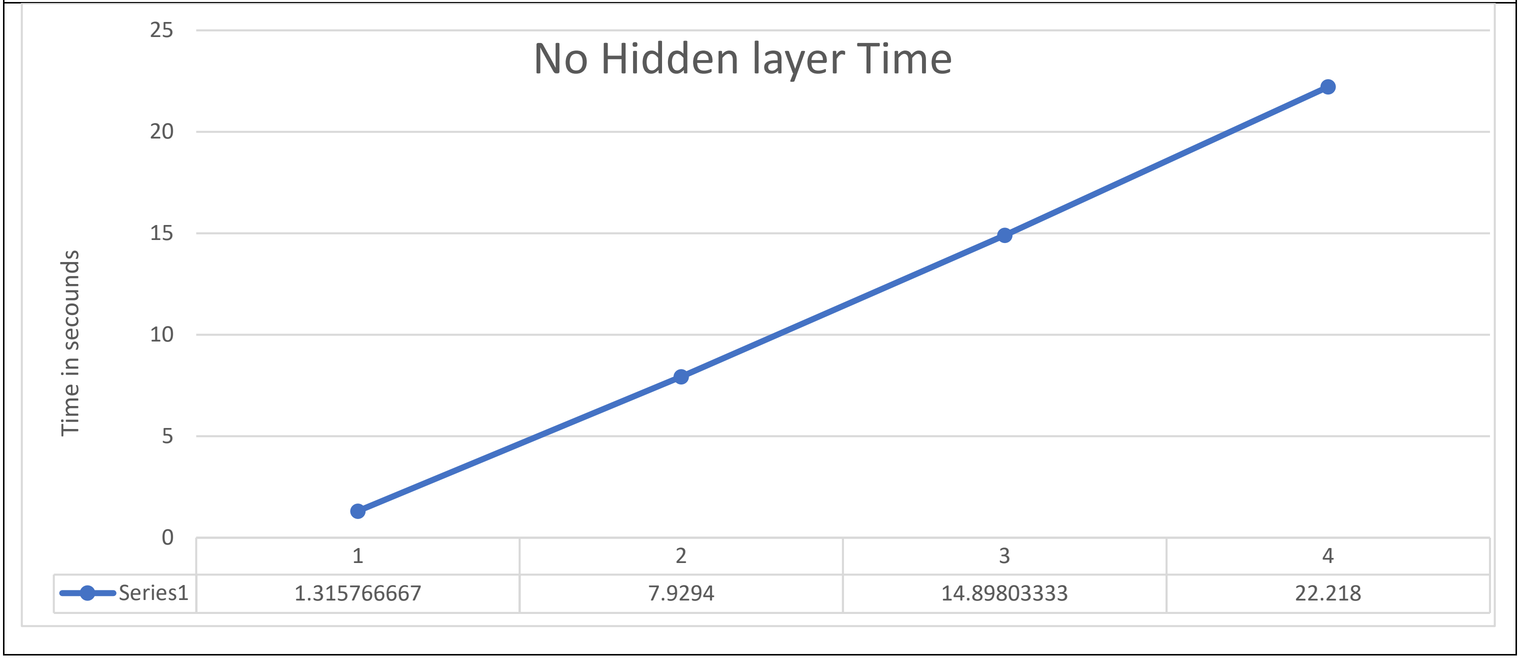


From the data the graphs, increasing the number of neurons will cause a stagnation after a certain point (20 neurons). There is an increase in time (though only like 10th of second) as you increase the number of neurons.

Performance – Number Hidden of hidden layer

After some investigation I discovered a source that said the optimal hidden layer would be one for most problems, however I still wanted to test it out. I tested from 1 to 4 hidden layers. I decide to test this last as I knew that increasing the layers would increase the total time. I ran this with all the same number of neurons.





Increasing the number of layers really did not increase the accuracy that much. It increased by 0.5% but increased the total time by 6 amounts. But the accuracy started to drop off and time really increased as layers increased. Ultimately, I felt that that was too much time to spend for just a 0.5% increase.

I did do another test using different number of neurons, but it didn't make much of a change to the overall accuracy.

Performance – Initial Verdict

The final values used for MLP were Sigmoid activation, a learning rate of 0.0225 a max iteration of 300, 25 neurons (a good balance of consistency and speed), a hidden layer size of 1, input size of 2 and an output size of 1.

These values gave the following results:

Classifier of iteration 0. Accuracy 97.94%, coverage 100.00%

Iteration 0, removed 1698 instances, instances left 0

Overall stats at iteration 0. Accuracy 97.94%, error rate 2.06%, not classified 0.00%

Accuracy 97.89%, Error rate 2.11%, not classified 0.00%, Time Taken: 3.32 Seconds

After doing a parameter of sweep of values close to the optimal value that I couldn't get another value more optimal than that.

Big Parameter sweep.

After these tests I wanted to make sure that I had not missed a route of optimization, I decide to run a big parameter sweep to make sure that I had been optimizing in the right direction

Actions	GAUSSIAN, RAMP, SIGMOID, SGN, SIN, STEP,
Neurons	5, 10, 15 , 20 ,25 ,30 ,35 ,40 , 45 ,50
Output	1
Learning rates	1.00E-04, 0.001, 0.0025 , 0.005 , 0.01, 0.025, 0.05 , 0.1 , 0.25, 0.5 , 1
MaxIterations	100,150,200, 250 , 300 ,350 ,400 , 450 ,500 ,550,600,700,750,800,850
Hidden layers	1

I am not looking at Linear , Log and Trapezoid becasuse they have only achived a accuarcy of 50% becasuse they arnt suited for this task. I have also removed Tanh as it keeps achiving a low accuarcy. Before I did this I tested how multipale hidden layers effect each action but found that they didn't really effect accuracy to much but caused a major increase in time. I also tested Max ints and found much like sigmoid each action overfitted with high max ints. I also discovered that Learning rates really effected each activation diffrently.

From running the pramater sweep here are my top 5 best results

name	neurons	output	learning	iterations	average	Error	time
SIGMOID	20	1	0.01	850	98.42105	1.578947	6.2904
SIGMOID	35	1	0.01	850	98.21053	1.789474	11.236
SIGMOID	30	1	0.01	800	98	2	8.9696
SIGMOID	35	1	0.01	800	98	2	10.4872
SIGMOID	30	1	0.025	300	97.89474	2.105263	3.4074

From these results its safe to say that SIGMOID was the correct activation function, its also furthuer supports that the optimal neurons is around 25. However what I missed was the iterations, I passed over 800. I did find that iterations of 300 which gave a good value (most the values after the top 5 are around 800 or 300 iterations.

The first value in this graph that wasn't Sigmoid was Gaussian , ranked 164 (97.21% accuracy).

The other activation functions didn't have a value appear until after 1000+

I wanted to do a sweep around this result to see if I could optimise the result further. This is my final value.

Actiavition	Sigmoid
LearningRates	0.008, 0.009 ,0.01 , 0.011 ,0.012
Hidden layer	19,20,21,22,23,24,25,26,27,28,29
Max Its	830 ,840 ,850 , 860 , 870

From this I got the following I got the following. This is my final value.

Activation	Neurons	out	learning	iterations	Average	Error	Time
SIGMOID	22	1	0.01	830	98.94736842	1.052632	5.533

Performance

I ran the optimal values again and got these results.

```
Classifier of iteration 0. Accuracy 97.59%, coverage 100.00%
Iteration 0, removed 1698 instances, instances left 0
Overall stats at iteration 0. Accuracy 97.59%, error rate
2.41%, not classified 0.00%

Final classifier cl0:
Stats on test data
Accuracy 98.42%, error rate 1.58%, not classified 0.00%

Total time: 5.75
```

Comparison on performance:

	GA	NN
Test Accuracy	98.42%	98.95
Test Error	1.58	1.052
Time	8.293	5.533
Final iteration	21	0
Final iteration Accuracy	100%	97.59

I have not shown the yang-yang diagrams as I believe that both the graphs would be too similar to distinguish as the accuracy is almost Identical (also I couldn't make decent enough graphs).

A genetic algorithm is a search technique that is used to find an exact or approximate solution to optimize a problem. If there is a problem where you want to find the worth of a potential solution that a genetic algorithm is probably the best.

A neural network are non-linear statistical models are used to find a relationship between the inputs and outputs/find patterns in data. NN can "learn" to classify objects that it has never seen before. For example, pieces on a graph.

Just on definition allow it, this suggests that NN would be the best out of the two as the whole point of this machine learning is to predict what color a value should be.

GA optimize – NN recognizes. [14]

Test Accuracy –

From the optimal values I have discovered, NN got the best value from the test data of over half a percent. Considering that near the end of each of the optimization processes getting a slight increase in percentage was difficult to do I find this is impressive. Admittedly I did spend a bit more time On NN and you must take into consideration that there is a little bit of randomness in each algorithm learning.

Time –

NN took the smallest amount of time (almost half the time of GA). I will admit that though I was trying to find optimal values which had an increase in accuracy I did try and optimize for time as well (I didn't have too long to run the tests). Something that was interesting when I was optimizing is that normally when a GA has to run for a long time then it's more likely its going to get a really good accuracy as it suggest that it has a high prediction processes, whereas long run time NN would normally get worst result because it is overfitting (not getting a good prediction values). The more the learning algorithm fits the data then its power to predict different sets of data is greatly reduced. However, in MLP the long run time Is probably because the parameter values passed in are ones that are often going to produce low accuracy (parameters value in the wrong area).

Iteration –

How GA and NN handle their classification are very different.

NN uses one iteration and returns a selection a weight back. The weights are used to calculate how optimal the algorithm is and the errors. Weights are the lines between nodes that multiplate how data flows throughout the NN. Weights with a higher values will have a greater magnitude over the other weights it connected to.

On each iteration GA creates a circle of a certain color and a certain radius size and plots them on a graph. The Circle is used to cover the instances in the training data set. Once all the instances are classified then the classifier is finished.

One issue that I found that GA is that there are some routes of optimization that will give you an 0.3 ish increase in accuracy but would increase the overall iterations (the highest I found was 34).

Final iteration –

The final accuary of both these test where high however GA got 100% accurate final iteration (and one which is 2% higher than that the NN), This suggest that the GA would poteinally be able to get a better test accuarcy if I had more time to sweep through more optimal values as the classifer was able to 100% predict the location of the dots.

Optimizing

I found that both algorithms where very easy to opmtise to get a relativity high accuracy, I did not feel like any of the parameters where waited on another parameter value allowing me to quite thoroughly tests to get an optimal value. They were both (after a little bit of tweaking) very easy to set up automated tests and data collection methods. One thing I did find is that in both algorithms is that there are many ways to optimize the code that will give you a very small increase in average accuracy however it would cause a large increase in time. This suggested they will both be optimal for a range of different problems as they both can be tweaked in many ways to set up take trade off time for accuracy.

Systematic sweeps

If you only had the option to do systematic sweeps, then GA is a lot better. I noticed in NN that there was a lot of hidden peaks and optimal values that where surrounded by un-optimal values. This meant that I had to go do very precise sweeps to find those values. Whereas in GA I could cast a large net and get a pretty good idea of where an optimal value should be (however I did spend more time on NN so it's possible that I missed something).

Big Parameter sweeps

If you had the time to do a brute force of a combination of parameters, then both are equally fine.

Overall reflection:

One the problems I had whilst doing this coursework was that I did have enough time to really dig into getting really refined values. Apart of this was because I had to balance a lot of other course works, another part was that I was waiting for feedback from coursework one. This coursework "flows" in a similar way to coursework one so I did not want to start the coursework until I knew that my method of optimization was "correct".

Another Issue I had was trying to change the knowledge representation for the genetic algorithm. I understand how they would work and why you would do it, it just could not get it to work in the code.

As this is kind of a follow on from course work 1 I did find it a bit easier.

Sources – I didn’t have time to do a proper Havard refencing

[1] https://www.educba.com/what-is-neural-networks/

[2] https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/

[3] https://skymind.ai/wiki/multilayer-perceptron

[4] https://en.wikipedia.org/wiki/Multilayer_perceptron

[5] https://machinelearningmastery.com/neural-networks-crash-course/

[6] https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/

[7] https://sefiks.com/2018/01/07/sinc-as-a-neural-networks-activation-function/

[8] https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f

[9] https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw

[10] https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

[11] Lecture notes

[12] https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a

[13] https://www.quora.com/What-does-weight-mean-in-terms-of-neural-networks

[14] https://stackoverflow.com/questions/1402370/when-should-i-use-genetic-algorithms-as-opposed-to-neural-networks