COSC420 Assignment 1 Report

Introduction

In this assignment I have made a Neural Network in Java that learns to map input to output by changing weights and bias from input to hidden layers and then hidden to output layers over a number of epochs. The weight changes are affected by a learning constant, momentum constant and error criterion. The number of hidden nodes in the hidden layer affects the time it takes for the neural network to learn.

In this report I have looked at the how the different parameters for the neural network affect the time it takes for the neural network to learn as well as how it learns for different tasks.

I have designed the Neural Network as such; there are three classes, the Main, which takes in some arguments from the command line and runs the neural network, the Neural Network which does the bulk of the work as it goes through many epochs to learn a specific task, and the Neural Node class, which acts in the place of a Node for the network, the Neural Network uses this final class to create the network.

Background

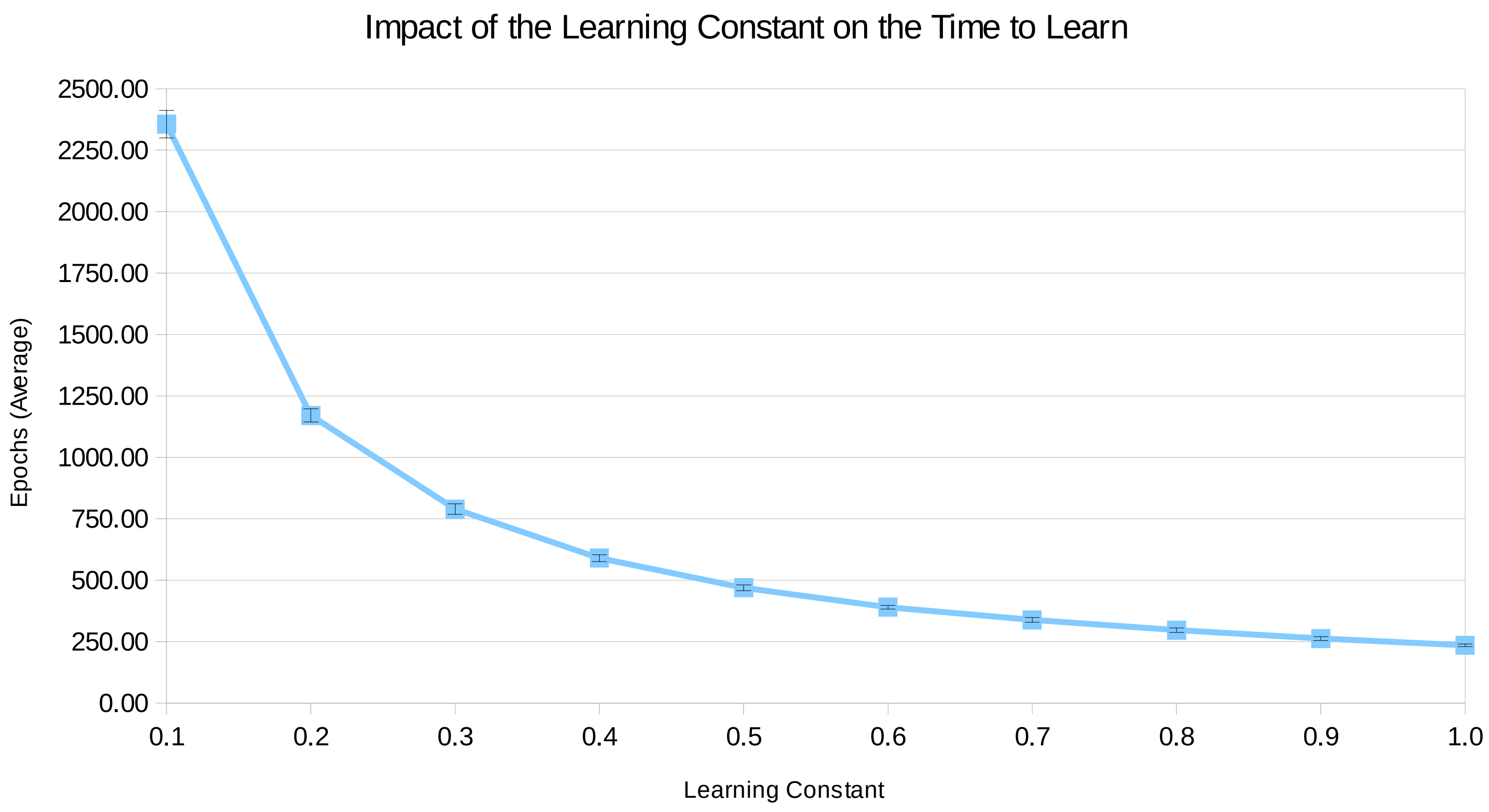
I have decided to explore how the different parameter settings affect the learning process, and how well or quickly does the network take to learn different kinds of tasks. For how the different parameter settings affecting the learning process, I will look at the learning constant, momentum constant and the number of hidden nodes in the hidden layer of the network on the XOR data set. For the different kinds of tasks I will have the network tested on 3 and 4 Bit Parity, Encoder, Iris and XOR, data sets from the COSC420 website.

Topic 1: Parameter Settings – Changing the Learning Constant

For this I will be testing the network on different learning constants with a range of 0.1 – 1.0 inclusive. This will be tested on the XOR data set, with a 2:2:1 network, the momentum constant will be 0.5 and the error criterion will be 0.001 through out the tests. For each learning constant, the program was ran 30 times, and the average, standard deviation, minimum and maximum are shown in the table below.

Results:

Learning Constant	Average	Standard Deviation	Max	Min
0.1	2355.67	56.26	2475	2233
0.2	1170.60	26.77	1218	1125
0.3	789.23	20.98	841	752
0.4	589.83	13.76	617	565
0.5	469.10	11.99	490	445
0.6	390.03	7.44	407	380
0.7	338.37	9.85	358	323
0.8	296.60	9.23	314	283
0.9	262.33	7.65	278	248
1.0	234.97	5.51	246	224



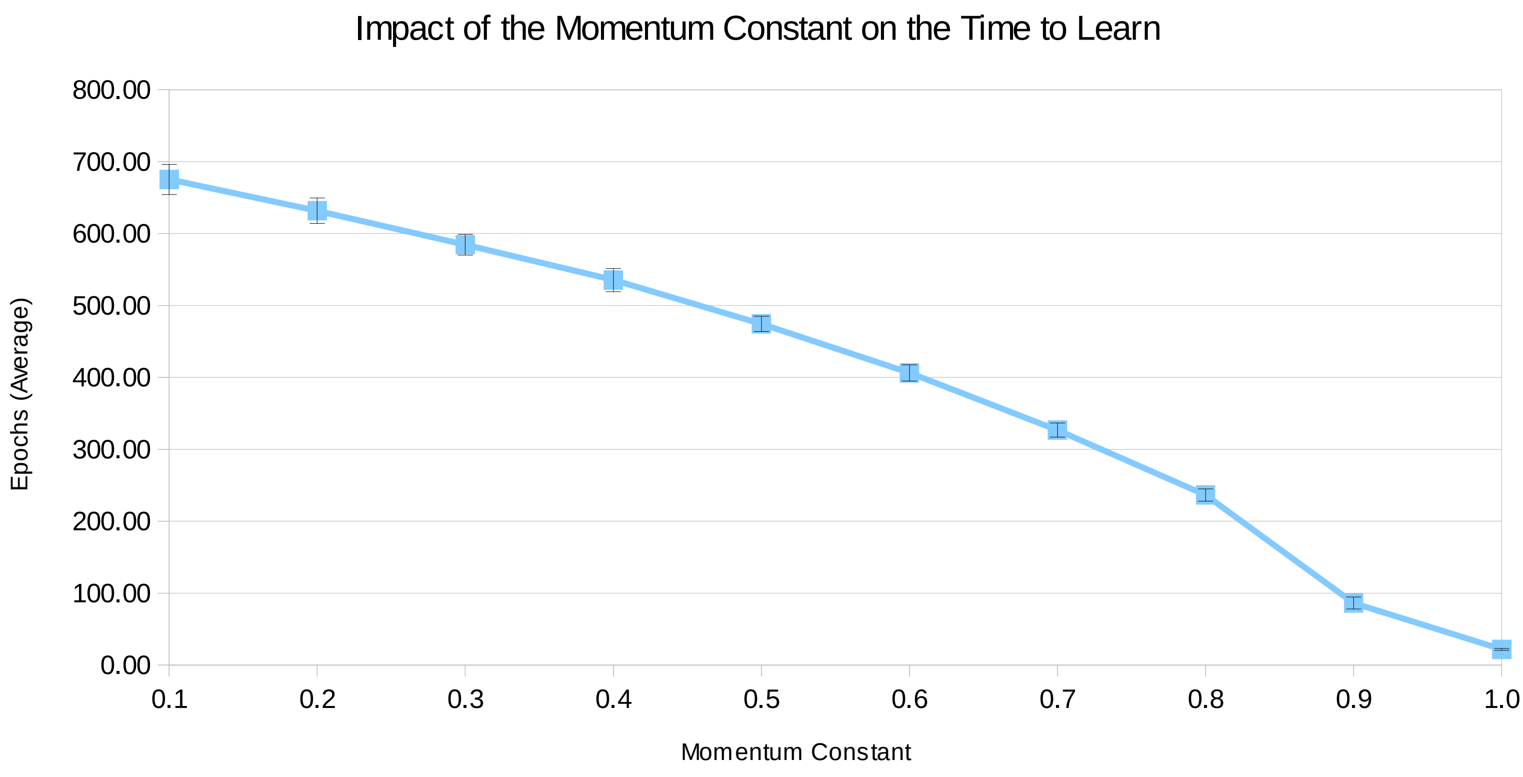
The graph shows that the learning constant has a logarithmic effect on the number of epochs it takes for the neural network to learn on the XOR data set. For a lower learning constant the range of epochs is far greater then a higher learning constant, ie the standard deviation of learning constant 0.2 is greater then that of a learning constant 0.8. It shows that the neural network is getting to the correct output faster and with more accuracy. The averages start to even out around the learning constant 0.6, this is when having a larger learning constant has less of an affect on the time it takes for the neural network to learn. From this, it shows that a larger learning constant does have a better affect on the time it takes for the neural network to learn, by making the network learn faster and making the difference between epochs less.

Topic 2: Parameter Settings – Changing the Momentum Constant

For this I will be testing the network on the different momentum constants with a range of 0.1 – 1.0 inclusive. This will be tested on the XOR data set, with a 2:2:1 network, the learning constant will be 0.5 and the error criterion will be 0.001 through out the tests. For each momentum constant, the program was ran 30 times, and the average, standard deviation, minimum and maximum are shown in the table below.

Results:

Momentum Constant	Average	Standard Deviation	Max	Min
0.1	675.27	21.01	748	642
0.2	631.73	17.69	672	607
0.3	584.50	14.37	630	563
0.4	535.50	16.09	570	511
0.5	474.37	10.54	504	453
0.6	406.30	11.39	429	388
0.7	326.73	9.91	350	309
0.8	236.47	8.62	253	219
0.9	86.30	8.37	105	73
1.0	21.83	1.34	25	20

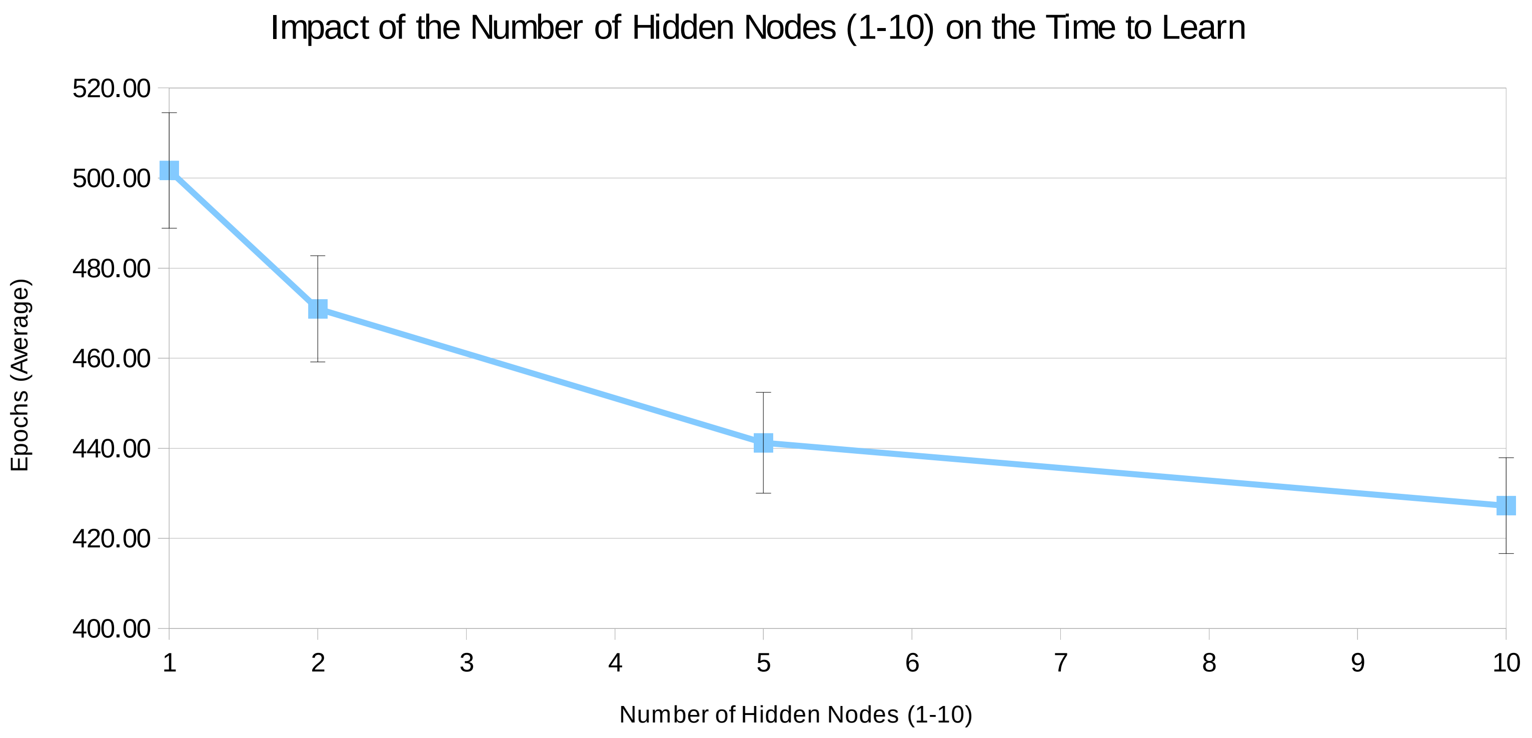


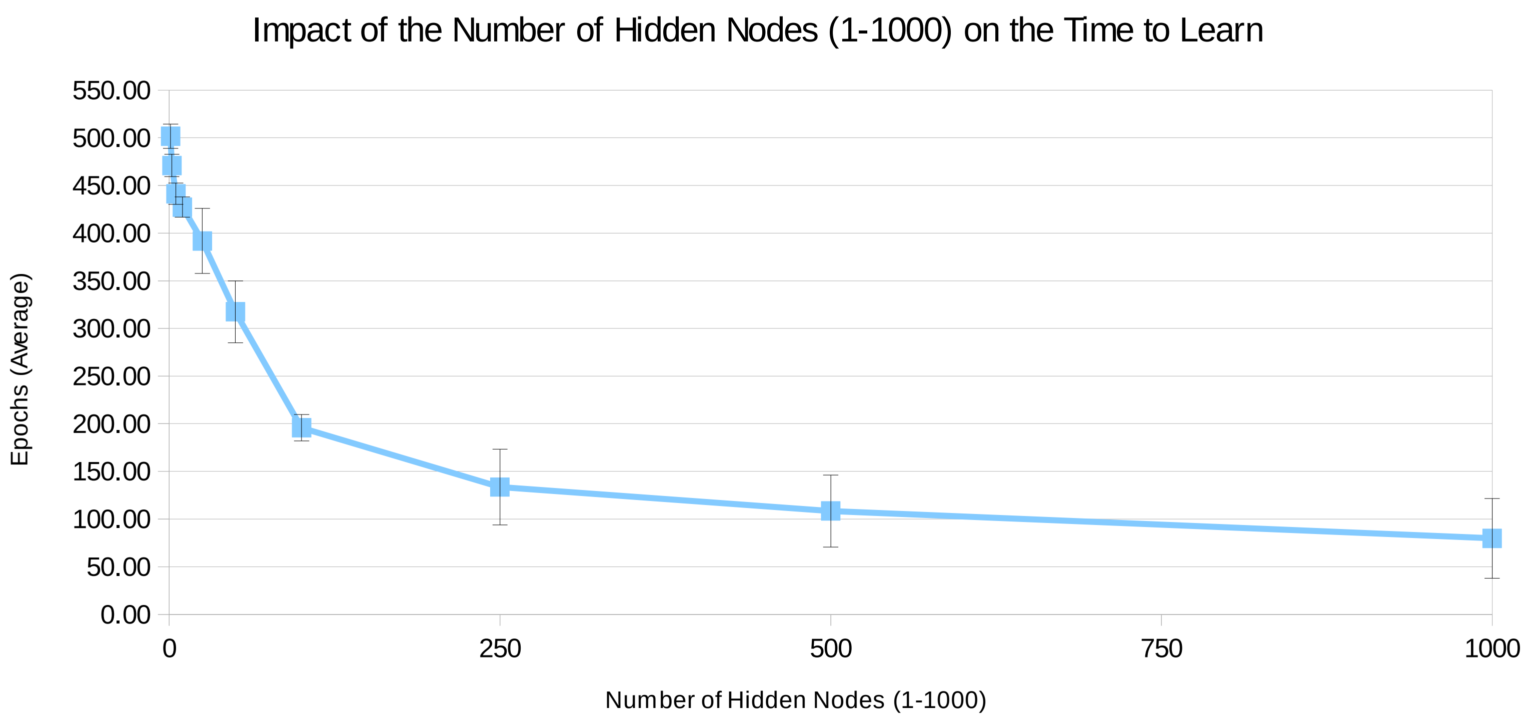
The graph shows that the momentum constant has an effect of making the number of epochs linear, between 0.1 and 0.8. This wasn't something that I considered would be the case, as I thought the momentum constant would make the epochs more random and sporadic, because of the fact that the momentum constant would be making larger changes to the weights to get the neural network learning faster. In all cases, having a larger momentum constant is better then having a smaller momentum constant, as it is always faster and better then having a lower momentum constant.

Topic 3: Parameter Settings – Changing the Number of Hidden Nodes
For this I will be testing the network on a different number of hidden nodes, values being 1, 2, 5, 10, 25, 50, 100, 250, 500, and 1000. This will be tested on the XOR data set, with a 2:n:1 (n being the number of hidden nodes) network, the learning and momentum constants will both be 0.5 and the error criterion will be 0.001 through out the tests. For each number of hidden nodes, the program was ran 30 times, and the average, standard deviation, minimum and maximum are shown in the table below.

Results:

No. of Hidden Nodes	Average	Standard Deviation	Max	Min
1	501.70	12.80	527	482
2	470.97	11.78	491	448
5	441.23	11.19	471	422
10	427.27	10.64	453	407
25	391.80	34.08	515	353
50	317.43	32.51	412	279
100	195.80	13.82	232	173
250	133.60	39.66	163	2
500	108.40	37.72	147	2
1000	79.70	41.88	165	2

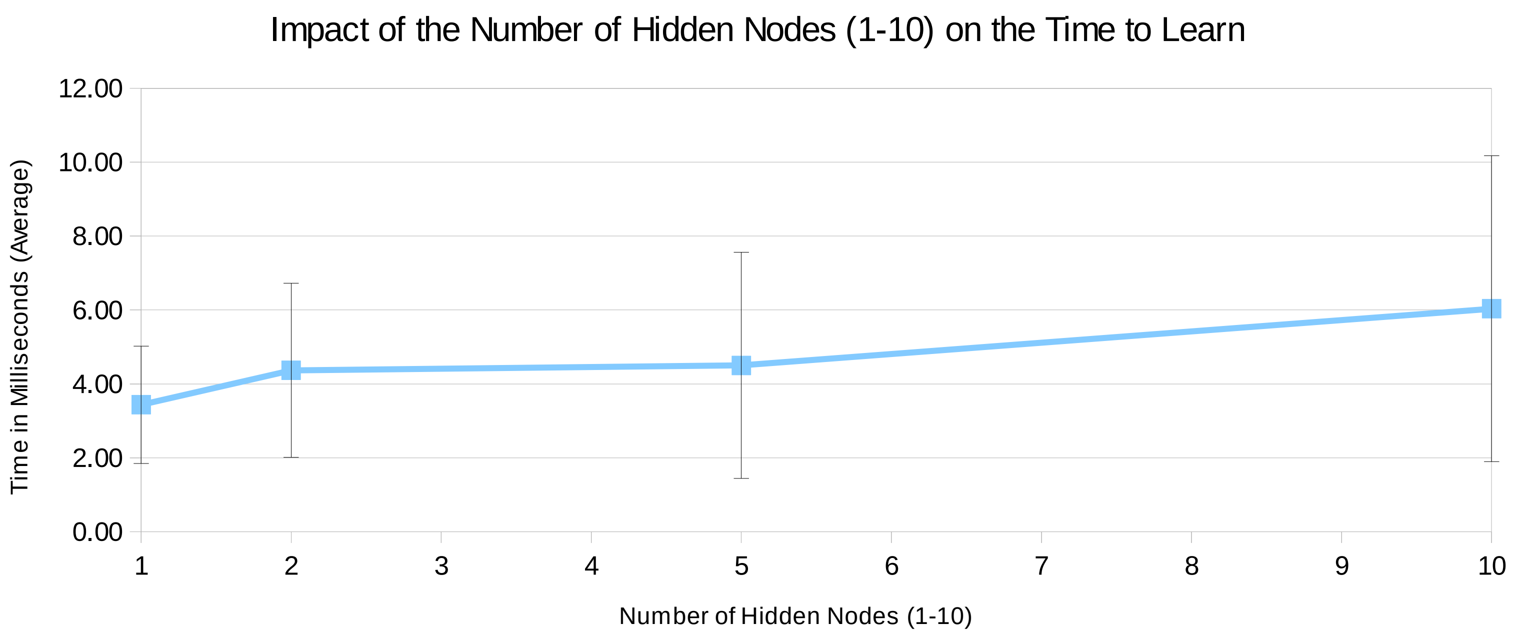


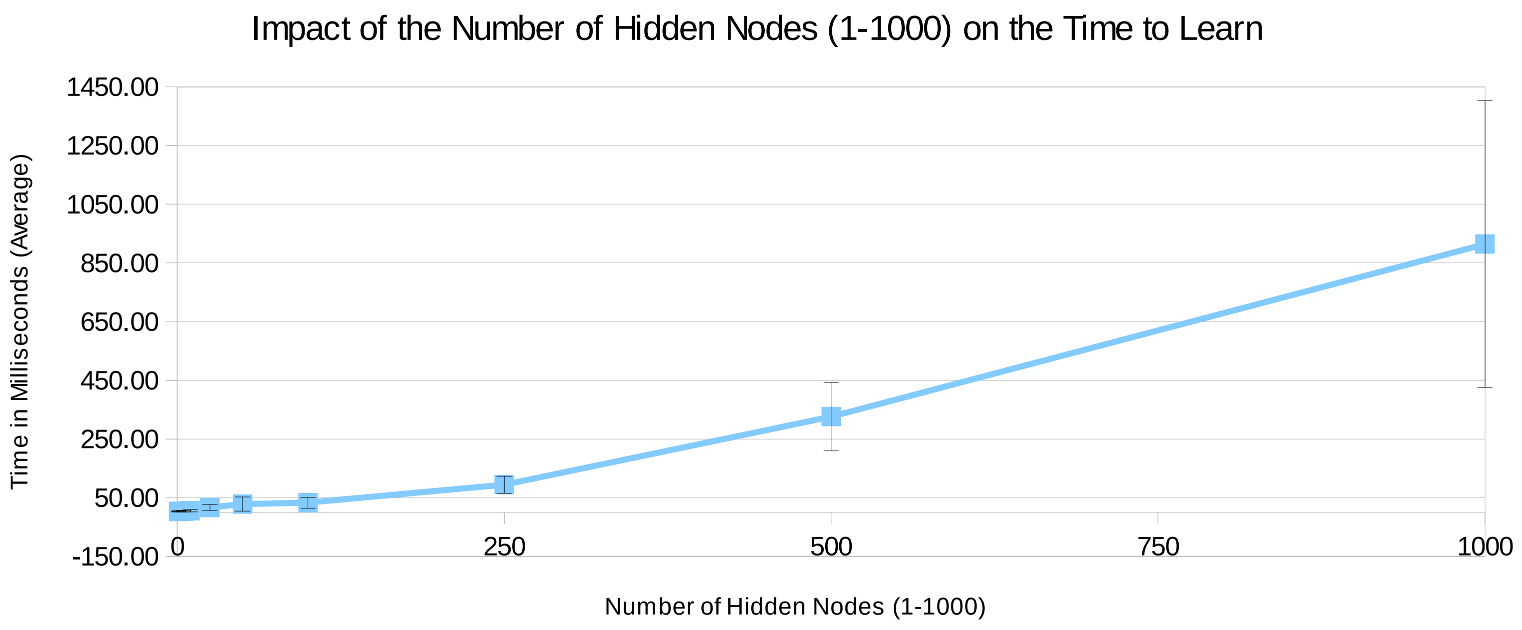


I have generated two graphs for this test, the first is only looking at the number of hidden nodes between 1 and 10, while the second looks at all the hidden nodes between 1 and 1000. The reason being it is hard to see the changes at the start of graph two. What this test showed is that having more hidden nodes can speed up the training of the neural network, but it would take up more time doing weight changes because of the number of hidden nodes. It shows when the number of hidden nodes is increased to 25 the range of different epochs is quite larger then the range of different epochs when the number of hidden nodes it set to 10 or less. The neural network struggles to get to the correct output when the number of hidden nodes is large. Another thing I experienced is that the neural network took longer to learn in time, a couple seconds longer. So I measured the neural network in milliseconds. The tables and graphs below were generated by the same values as the tables and graphs above.

Results:

No. of Hidden Nodes	Average	Standard Deviation	Max	Min
1	3.43	1.59	9	2
2	4.37	2.36	12	1
5	4.50	3.06	18	2
10	6.03	4.14	24	3
25	16.93	10.53	43	5
50	28.67	24.30	72	11
100	33.10	18.47	101	22
250	94.80	29.19	184	5
500	326.50	116.49	500	7
1000	914.37	488.96	2051	23



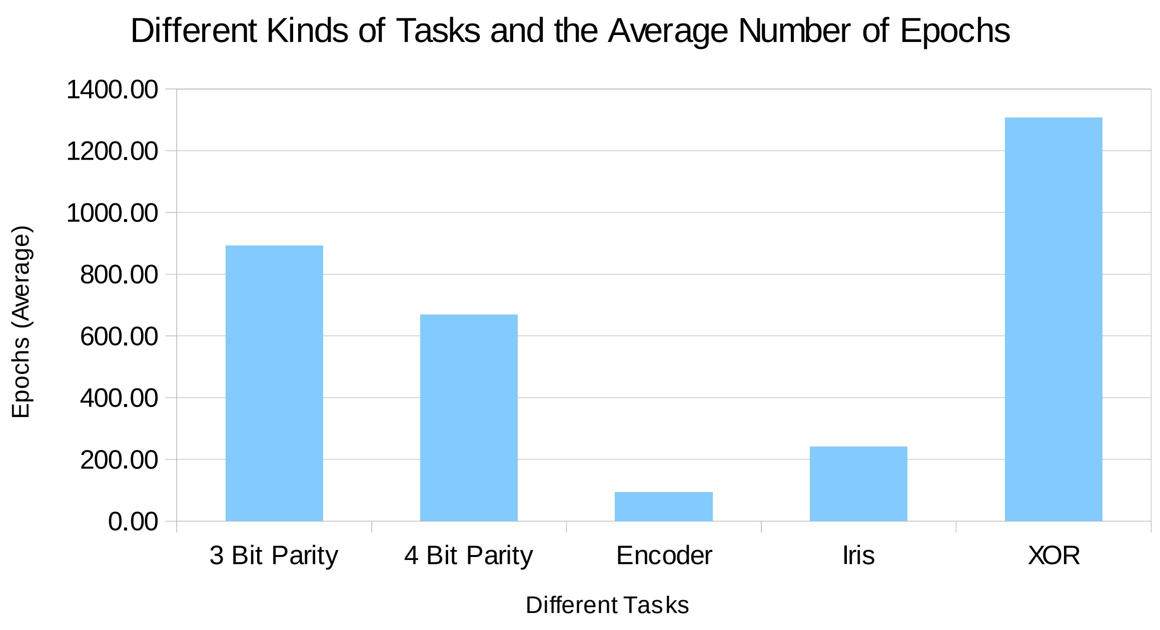


The graphs show that having more than 10 hidden nodes in the neural network makes the network run slower in terms of seconds, and the range of different time is quite large, ie for 50 hidden nodes, the range is between 11 and 72. It makes sense as there are more weights that need to be updated for each hidden node in the hidden layer. Again this shows that having more hidden nodes does not help the neural network learn faster, it just adds more unnecessary work.

Topic 4: How well or quickly the network takes to learn different kinds of tasks
The different tasks I am using for this experiment are 3 Bit Parity (3:3:1 network), 4 Bit Parity (4:4:1 network), Encoder (8:3:8 network), Iris (4:3:3 network), and XOR (2:2:1 network), where the learning and momentum constants are 0.7 and the error criterion is 0.0001. The program was ran 30 times for each of the tasks and the average, standard deviation, maximum and minimum are shown in the table below.

Results:

Different Tasks	Average	Standard Deviation	Max	Min
3 Bit Parity	892.23	14.64	925	866
4 Bit Parity	668.10	8.88	687	652
Encoder	94.43	0.94	96	92
Iris	241.27	0.52	242	240
XOR	1306.93	27.49	1362	1237



The above graph shows that for simple input and output that match, the neural network learns very fast, less than 100 epochs, and if there is a lot of input and output given to the network to learn, based on the iris data set, it will also learn pretty quick. For the other three tasks, 3 and 4 bit parity and XOR, the neural network has to take more time to learn, because these tasks only have one output node that has to map to multiple input nodes. Due to this fact, there is a greater difference of epochs for there respective task.

Discussion

From all these tests it shows that having a greater learning and momentum constant help by a significant amount in speeding up the training of the neural network. Which makes perfect sense as these values add to how large a change is made to a weight during every epoch. What was interesting is how the learning constant stops making significant changes to the number of epochs after the learning constant reaches 0.6 and larger. Initially I would have thought the differences would have been larger. Another thing that I confirmed with these tests was the fact that having more hidden nodes in the hidden layer didn't help speed up the network when learning. Running the neural network on different tasks and graphing them together shows how different some tasks are to each other. XOR being the most complicated thing the neural network has to learn compared to the other tasks that are shown.

Some future expansions to the code I would consider doing would be adding more hidden layers to the neural network, and allowing the number of hidden layers to be specified like how the number of input, hidden and output nodes are specified, and seeing the effect on how the network learns if it would increase or decrease the time. Though that would be complicated, it was hard enough just getting one hidden layer working for me. I would also like to try the neural network out on some more complicated data sets, and more real world data sets like the iris set, and see how the neural network holds up.

Appendix

The program was developed under Java version “1.8.0 update 45”.

I have included a MakeFile to compile the source code, all you need to do is type “make” in the same directory as the MakeFile and the source files.

If for some reason the MakeFile does not work, then you can compile the code with “javac *.java” in the directory of the source code.

To run the program, use “java Main”

The program can take two arguments, the first being a string made up of chars

e = Show the population error and number of epochs every 100 epochs

w = Show the weights of the hidden layer and output layer

a = Show the activation all nodes in the network, input, hidden and output

l = If you want to loop the program a certain number of times, this is when the second argument is used. The second argument should be an integer, and it represents the number of times the program will run.

Example usages:

```
java Main aw
java Main el 5
java Main wlea 100
```

The program will require that three text files be in the same directory as the executable, param.txt, in.txt and teach.txt. The param file specify the number of input nodes, hidden nodes, and output nodes, as well as the learning, momentum and error criterion constants. The in file is the activations for the input layer, and the teach file is the desired output for the neural network.