Neural Network

Olof Harrysson

January 2017

1 Resources

The algorithms were implemented in Python 3.5.1. The code was run on a 2,3 GHz Intel Core i7, MAC OSX Yosemite. The library Numpy was used to create the graphs presented. Information from lectures and the following websites influenced the project greatly.

`http://neuralnetworksanddeeplearning.com/`
`https://www.coursera.org/learn/machine-learning`

2 Structure

The network is built by an input layer, a hidden layer and an output layer. The number of nodes in each layer is adjustable. The network uses bias nodes for the input and hidden layer. The weights for the connections (including the bias nodes) are initialized to a random number close to 0. The sigmoidal function is used to transfer the nodes input to output, even in the last output layer.

The neural network used in this report has 12 input nodes, 10 hidden nodes and 1 output node. The learning rate is set to 0,1.

3 Setup

The Y values (solutions to our predictions) are normalized to values in the range [0,1]. This is done so it can be compared to the output nodes output which will be in the range [0,1]. Before training the network, the data is split up into training data and test data randomly.

4 Training

An outer loop is training the network a set amount of times called epochs, or until the max time parameter is reached. First, the network does a feed forward on all the input data. It calculates the difference of f(x) with y and begins the back propagation algorithm. After this the network uses saved variables to update it's weights and biases. The network is implemented to use matrix calculations in almost all cases.

5 Testing

After every epoch, the root square mean function is used to calculate the cost on the test data. The error in the last layer is calculated and scaled up by the same factor used to scale down the Y vector in the setup phase.

6 Results

The network was benched marked against a previously implemented genetic algorithm and a swarm optimization. The data is plotted in different plots with different scales to make it easier to understand the data.

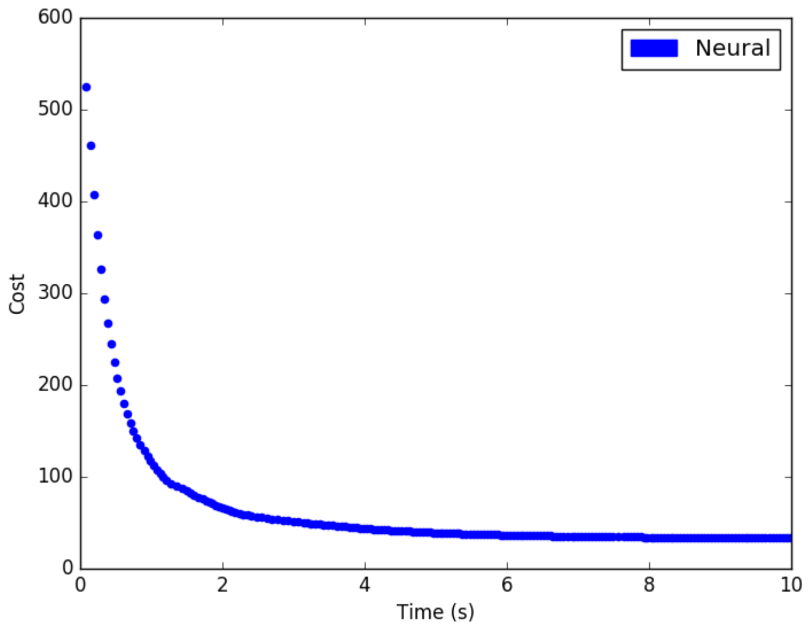


Figure 1:

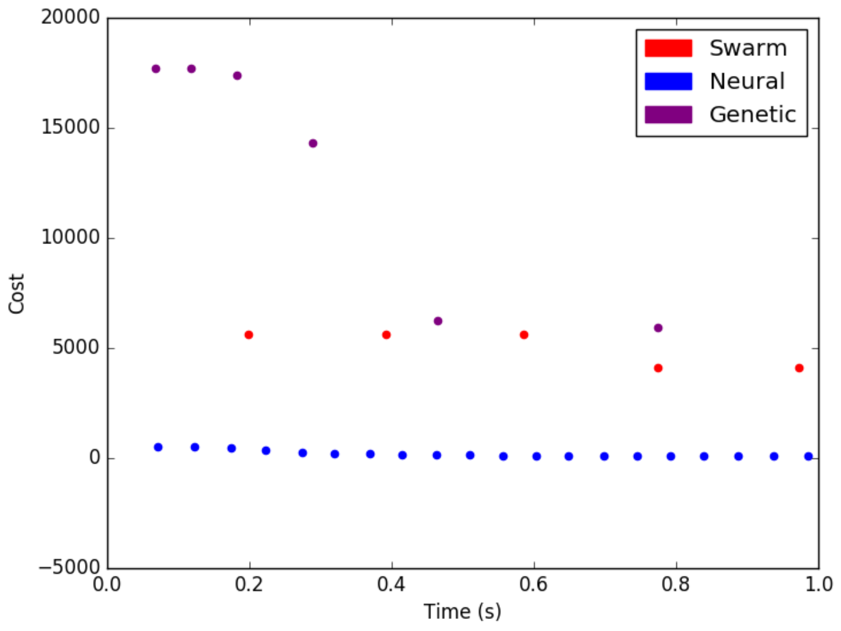

Swarm
Neural
Genetic
Cost
Time (s)
20000
15000
10000
5000
0
−5000
0.0
0.2
0.4
0.6
0.8
1.0

Figure 2:

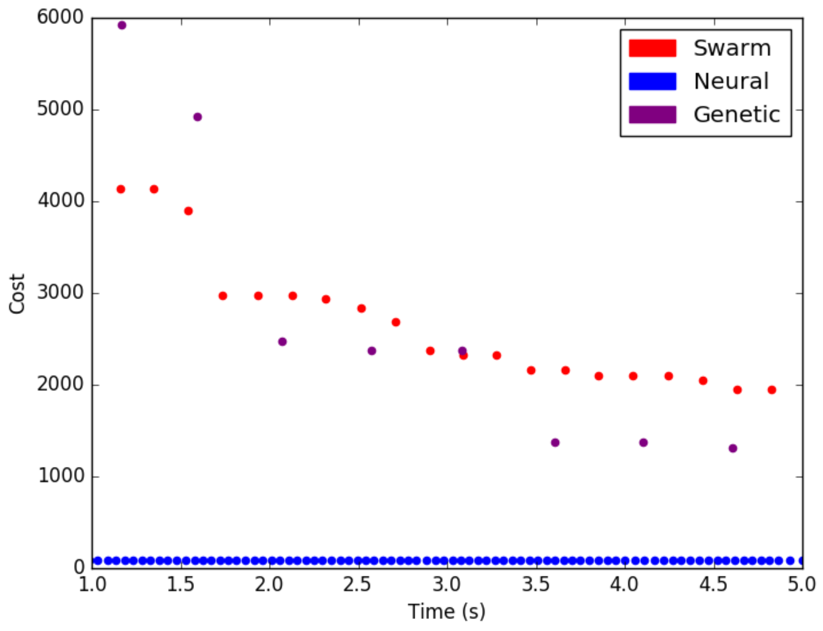

6000
5000
4000
3000
2000
1000
0
Cost
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Time (s)
Swarm
Neural
Genetic

Figure 3:

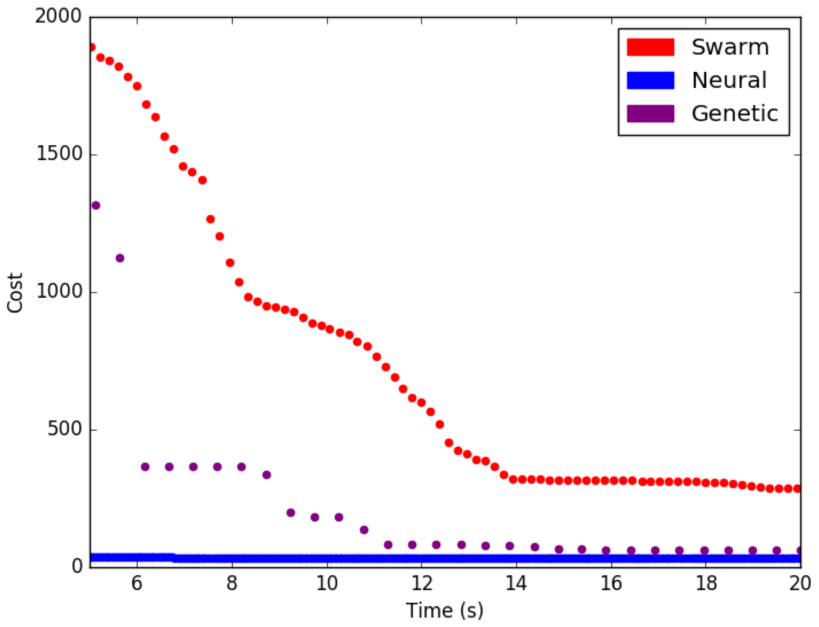
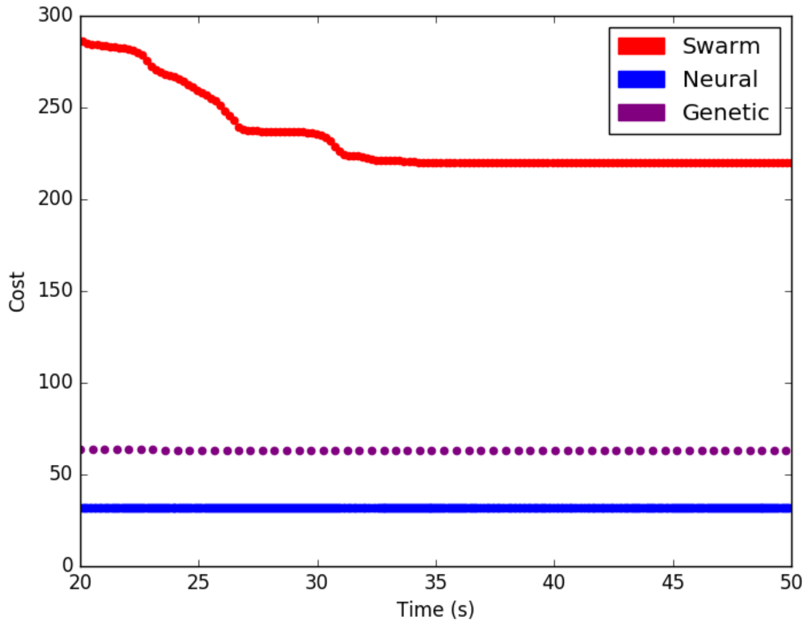

2000
1500
1000
500
0
Cost
6
8
10
12
14
16
18
20
Time (s)
Swarm
Neural
Genetic

Figure 4:



300
250
200
150
100
50
0
Cost
20
25
30
35
40
45
50
Time (s)
Swarm
Neural
Genetic

Figure 5:

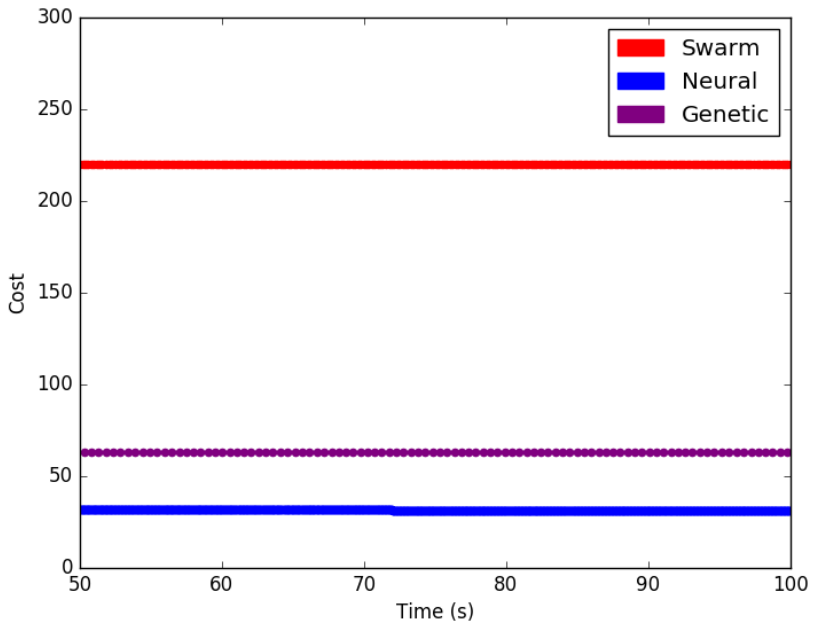

300
250
200
150
100
50
0
Cost
50
60
70
80
90
100
Time (s)
Swarm
Neural
Genetic

Figure 6:

7 Conclusion

The neural network outperforms the other algorithms by far. It was however much harder to implement than the other algorithms. If the network gets to run for a long time, say 100 seconds+, the cost will start to increase slightly. Perhaps this is due to over fitting the network to the training data. Originally, the program was normalizing the input data to have a mean of 0 and a standard derivation of 1. This was done to not saturate the sigmoidal function in the hidden layer. This idea was scrapped because when the network ran for a long time, the cost function increased more than if one were to use non normalized inputs.
Another idea was to use a technique called stochastic gradient descent to improve the time it took to train the algorithm. The technique randomly selects a part of the training data for every epoch making the calculations faster. After many epochs, the network will be trained similarly to a network where the stochastic aspect isn't used. There was no need to implement this as the training on the full training data was quick enough.