HERIOT-WATT UNIVERSITY

Biologically-Inspired Computation

COURSEWORK 1

Student
Mark ASKEW - H00153337
ma1017@hw.ac.uk

Contents

1 Introduction

The aim of this report is to give an overview of the neural network and genetic algorithm which were built to emulate mathematical functions. The results of running the different mathematical functions on the neural network will be graphed and then analysed to assess the effectiveness of the program.

1.1 Neural Network

1.1.1 Input layer

Each of the mathematical function inputs is mapped to a single node in the input layer. I decided to restrict the neural network to take inputs with a maximum dimension of 2 and so the input layer contains only 2 nodes. This unfortunately means that the program is restricted to dealing with inputs with a maximum dimension size of 2, however, utilising fewer nodes and weights in the structure can make the network easier to train (Zhang, Lu and Kwok, 2010). The neural network strucure used can therefore return more accurate results with smaller numbers of training iterations compared to networks which have more neurons and weights.

1.1.2 Hidden layer

Due to the complexity of some of the mathematical functions which the neural network is required to emulate, a hidden layer is required in order to successfully process the data. As there can be multiple hidden layers in the neural network architecture, it is important to select the number of layers which will lead to the best performance of the program.

As noted by Hornick in his paper on multilayer feedforward networks (Hornik et Al, 1989), multilayer feedforward networks with as few as one hidden layer are capable of approximating any continuous function provided that a sufficient number of hidden units are available. This means that if a neural network with 2 hidden layers can succesfully approximate a function, then a hidden layer with 1 hidden layer can also successfully approximate the same function. Although this point is true, it can be very difficult to approximate complex functions accurately using only a single layer and by

using additional hidden layers with limited numbers of units it has been found that better results can be returned more easily (Nielsen, 2016).

The more hidden layers that are used however, the greater the computation and training that is required to train the network and so it is important to find a balance. The final implentation of the neural network has two hidden layers of 5 nodes in order to gain succesful results but also keep the required computation and training to a reasonable amount. I decided to use a bias node in each of the hidden layers to help the network to scale its output values better (Nielsen, 2016).

1.1.3 Output layer

As the problem which the neural network is trying to solve is a regression problem, only one output node is required. This node is used to output all of the neural networks results.

1.1.4 Neural network structure

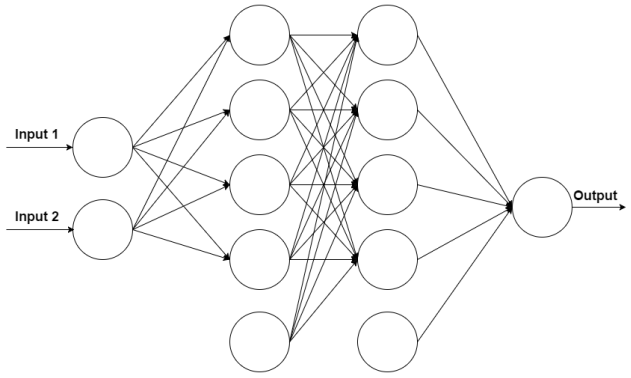


Figure 1: Neural network structure

The Structure that I decided to use had 2 input nodes, 2 hidden layers each with 4 normal nodes and one bias node, and an output layer with one node.

1.2 Genetic Algorithm

To start with a neural network is created with random weights, these weights are then passed into the genetic algorithm function. An initial population(first generation) of neural networks is created using these weights along with a random mutation in order to make them all unique. Each neural network is then run on a set of test data and the fitness of each network will be calculated. The weights of the fittest neural network in the generation will then be passed onto the next generation of neural networks for mutation. The fittest neural network of each population will continually be passed on to the next generation until the maximum number of generations has been reached. When the last generation is completed, the fittest neural network for all of the generations will have its outputs and attributes returned by the program.

1.2.1 Fitness function

When creating a fitness function for the genetic algorithm, I wanted to use a function which would work effectively for all of the mathematical functions being used. The fitness function selected uses the 'Cost' value that is returned by the neural network every time the program is run. This cost is essentially a measurement of how close the output values of the neural network were to the expected output values from the input.

$$1/2(\hat{Y} - Y)^2$$

Figure 2: Cost function of the Neural Network

In the above equation, $\hat{Y}$ equals the output of the network and Y equals the expected output. The cost values generated for all inputs to the program are added together to create a total cost.
Using this equation, if an output value is a lot bigger or smaller than the expected value - then a very large cost will be calculated due to the squaring. This will help find neural networks which return results which are all relatively close to all of the expected results and not close to some and nowhere near the others.

1.2.2 Mutation

After the population of the next generation has been created, the weights of each neural network need to be altered in order to create better solutions. The number of weights that are changed for each neural network are randomly selected. This allows for the agents of the population to be unique from one another. The probability of weight mutation can be altered by changing the mutation frequency variable.

The mutation strategy uses the cost of the neural network to deduce how large the changes to the weights need to be. If the cost is currently large, then the change in the weights also needs to be relatively large in order to get closer to the solution. If the cost is low and the neural network is close to emulating the mathematical function then only a small adjustment needs to be made to the weights in order to improve the neural network.

The difference between the output results of the neural network and the expected results is used to determine whether the weights of the neural network need to be increased or decreased. If the sum of all the output values of the network is greater than the sum of all of the expected values then the computed mutation value will be taken away from the selected weights, otherwise it will be added.

```
mutate = random.randint(1, mutationFrequency)
if (mutate == 1):
    if(difference > 0):
        nn.W2[n] = bestW2[n] -(learningRate) * cost
    else:
        nn.W2[n] = bestW2[n] + (learningRate) * cost
```

Figure 3: Mutation code of program

2 Experiment Setup

There are three main parameters that are used in the genetic algorithm:

- Learning rate - A values used to restrict the extent of the weight change made during mutation
- Population size - The number of neural networks which are created using the same weight inputs
- Generation size - The number of times that a best neural network is selected from a population

Generally, the higher the population and generation size are - the better the returned results will be. This is shown in the following two graphs.

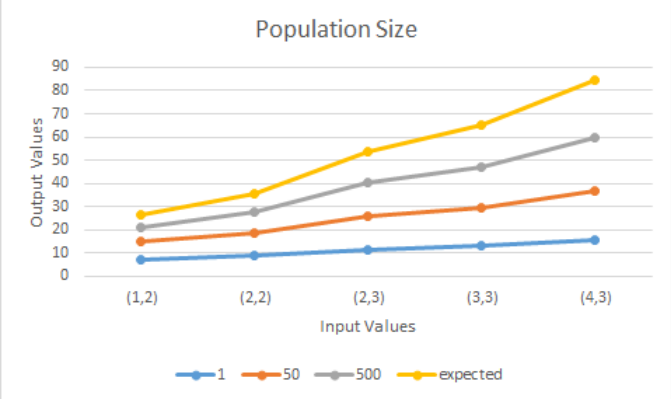


Figure 4: Result of running neural networks with different population sizes on a data set

The size of the best learning rate is consistently a very small value. The following graph shows that when the program uses a small learning rate it can adapt better to values in different ranges. The data in the graph has been scaled so that all of the data is readable.

I used a learning rate of 0.0001 as this learning rate is proven to work well with data in different ranges.
I decided to use a population size of 300 and a generation size of 300 for each of the experiments. These population and generation sizes are large

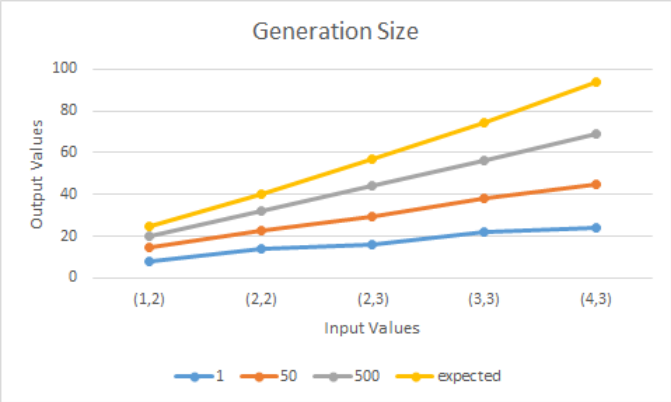


Figure 5: Result of running neural networks with different generation sizes on a data set

enough to return good results but are small enough that they won't take an excessive amount of time to run.

A data set of [(1,1), (1,2), (2,2). (2,3), (3,3)] was used as inputs for each of the mathematical functions.

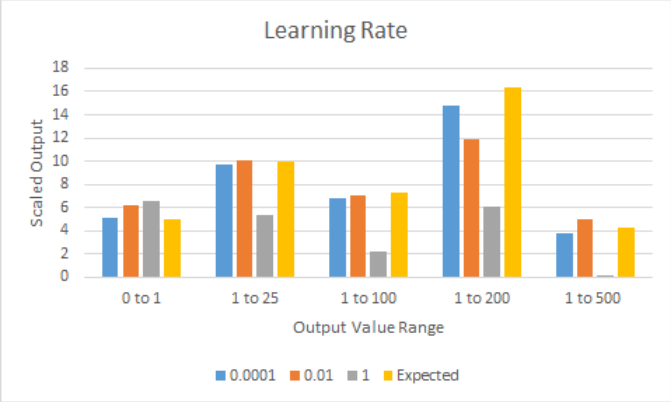
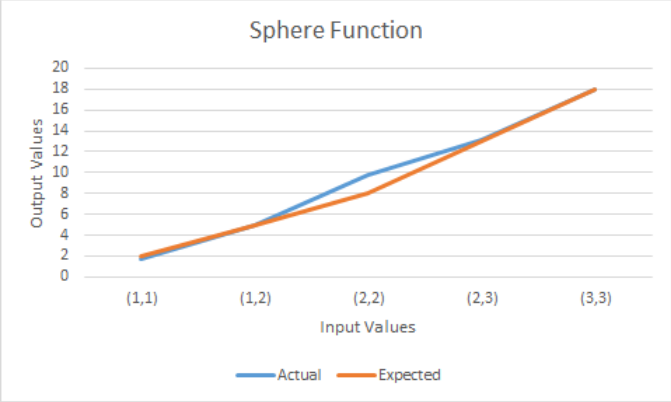


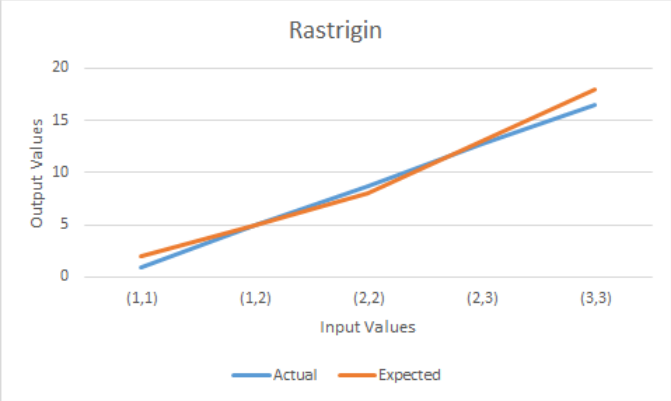
Figure 6: Result of running neural networks with a different learning rate on a data set

3 Results

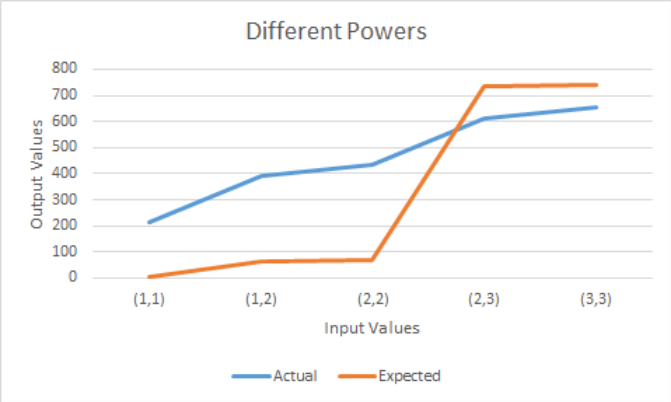
3.1 Sphere Function



3.2 Rastrigin Function



3.3 Different Powers Function



4 Discussion

The program creates very accurate results for the sphere function. I believe the reason for this is that the sphere function is relatively simple in nature and the inputs of the data set create outputs which are within a small range of numbers. These attributes make it easy for the network to approximate the function.

The results of the Rastrigin function were also quite accurate. The expected output values, like the sphere function, were all in a small range and this made it easy for the network to approximate the function.

The results from the program using the different powers function are quite innacurate. The different powers function is a more complex function than the sphere function and the output results are a lot more varied and are spread over a large range of values. The graph does show that the curvature ofthe expected and actual result lines ,which shows that the neural network is definitely adapting to the inputs but just not well enough.

5 Conclusion

To conclude, the program was successful when emulating relatively simple functions like the sphere function and the Rastrigin function but for functions that are more complex in nature - the results decreased in accuracy. I believe the reason for this is that fitness function and mutation strategy of the genetic algorithm don't allow the network to adjust its weights precisely enough to cater to individual input values. The fitness function and genetic algorithm just make imprecise changes to the weights of the neural network.

Github repository - https://github.com/Maskeww/bioCoursework1

6 References

Zhang, L., Lu, B. and Kwok, J. (2010). Advances in neural networks–ISNN 2010. 1st ed. Berlin: Springer.

Hornik, K., Stinchcombe, M. and White, H. (1989). Multilayer Feedforward Networks are Universal Approximators. 1st ed. Wien.

Nielsen, M. (2016). Neural Networks and Deep Learning [online] [Accessed 14 Nov. 2016].