NEURAL NETWORKS

Neural networks and Genetic algorithms to minimise a set of continuous functions

Carlos Alberto Chavez
Cac5@hw.ac.uk

Chavez, Carlos A
[Email address]

Introduction

The algorithm used in these experiments use a neural network combined with a genetic algorithm to minimise a set of functions defined by COCO (COmparing Continuous Optimisers). The neural network is trained with both a genetic algorithm and a backpropagation method to compare approaches. The parameters on the execution can be changed to evaluate each of the approaches, these include a bit to check if the genetic algorithm is going to be used, number of output neurons, mutation rate, learning rate, momentum, and number of executions.

Code for this work was generated using Matlab for testing and learning purposes. Since performance is of major importance, a programming language focused on this matter should be used, for example c++ or c# (same code was tested using c# and Matlab with a simple example and the time taken to run 1000 generations varies from seconds to minutes). The solution was created using an object-oriented approach, to facilitate the understanding of the functionality and testing. For performance improvement, lists of objects were treated as fixed sized matrixes. Code is well documented, to understand from scratch or to modify sections as needed for future references. Code is stored in the github environment in the following URL:

https://github.com/cchavez24/NeuralNetworks

Code must be run using the class “exampleexperiment.m”. A test of the neural network algorithm combined with the genetic algorithm is done using a fixed expected value in the class “experiment_NN_TEST.m”. Classes and objects are stored within the same path.

Neural network algorithm

Creates a neural network following a topology. Every topology follow the same mutation rate, learning rate, momentum, number of generations, but code can be modified in a simple way for every neural network to have their own values for these properties. Learning rate and momentum are only used when backpropagation is used to train the neural network, if this approach is not used (for example using a genetic algorithm) these values can be zeros, they will not be considered. Mutation rate, learning rate and momentum values are normalized from 0 to 1 (including all in between). These values can be changed within 2 classes: “MY_OPTIMIZER.m” and “Experiment_NN_TEST.m”.

Each topology must follow these rules and assumptions:

- Number of inputs are the same as the number of neurons in first layer (input layer).
- There is at least an input layer, one hidden layer, and an output layer.

- The output layer has only 1 neuron. Code was generated in such a way that the output layer could have more neurons, but this is not completely tested in the scope of this work.
- Each layer can have at least 2 neurons and at most 10, excluding input and output layer. (Code was tested using a 1 x 1 x 1 topology, but not considered for any metrics).
- Each layer includes a bias except the last layer. Bias neurons have a fixed output value of 1, random weights at their initialization, and no other neurons outputs towards them (in other words, no inputs for every bias irrespective of the layer). Biases are not included in the topology, but are created in the initialization of the neural network automatically.

Creates the topology by using random numbers for the number of layers, and number of neurons in each layer. In each layer, creates a matrix of neurons, and an equally sized matrix of output weights, and output weights deltas. These values are going to be used in the feedforward and backpropagation methods.

Feedforward to obtain the output value of each neural network.

Hyperbolic tangent activation function to use values between -1 and 1. Could use sigmoid function which has values from 0 to 1 but would only add values, while hyperbolic tangent would also include subtraction. Code for the sigmoid function was used and implemented, but must be changed manually in the call for the activation function in the feedforward method.

Backpropagation approach

Weights are modified with the error found in the last layer and back propagating a delta change in weights. Error is calculated by the mean square error which is the square of the delta of the value calculated with the expected value. Could use simple delta average of error or mean cross entropy error, but these approaches were not implemented in code. This approach is only used for testing purposes and for comparing with the genetic algorithm approach.

Genetic Algorithm approach

The main use of the program is based on the combination of a neural network and a genetic algorithm. This approach not only evolves the weights in each neural network created but also evolves its topology. The principle follows these steps:

- Generates a random population of elements (neural networks) between 5 and 20.
- Evaluate fitness of every element. Fitness function in this case is the absolute value of the difference between the expected value minus the calculated value (delta).

- Take the best individual of the population to use it as a parent for the next generation. This parent is chosen by the element in the population that has the smallest error. Every element is combined with the best element to reduce the error and keeping a best parent for future generations. The best parent was later replaced by the next generation's best element, that is the element with the new least error.
- Generate child elements using the best individuals chosen and cross validation between them. The cross validation is simply the sum of both weights of both parents divided by 2. The result is the absolute value of this operation.
- Mutate the child element depending on a mutation rate (probability of it mutating after being crossvalidated).
- Replace member of the population with mutated child.
- Iterate as many generations and as many neural networks there are.

Conclusions

Use high numbers of training data to avoid overfitting, in this case using a high number of inputs for the evaluated equations to assert that the topology is working for most of them. Also, use various numbers of different topologies to train the neural network.

At the end of the training, the best topology is chosen by the one that had least error at the end of the iterations. The solution includes the number of layers, number of neurons in each layer, and weights for every output of neurons including the bias. For higher complexity problems (functions with higher dimensions) the solutions contain topologies with higher number of layers and higher number of neurons in each layer. These topologies let the error be reduced in the least generations possible. By using backpropagation and a genetic algorithm approach to modify the weights, a faster solution was found when the genetic algorithm was used, because the evolution happened faster, while backpropagating had to be done for all layers and with a high number of generations. Evolving the weights with crossover with the best weights was the best solution.