

---

# Apply genetic algorithm to the learning phase of a neural network

---

**Sergi Perez**

Department of Mechanical and Aerospace Engineering  
University of California, Irvine  
sperez1@uci.edu

## Abstract

Natural networks have been used during several years to solve classification problems. The performance of a neural network depends directly on the design of the hidden layers, and in the calculation of the weights that connect the different nodes. On this project, the structure of the hidden layer is not modified, as the interest lies only on the calculation of the weights of the system. In order to obtain a feasible result, the weights of the neural network are calculated due a function cost. A genetic algorithm approach is presented and compared with the gradient descent in failure rate and time to obtain a solution.

## 1 Introduction

Neural networks is a computational model based on the neural connections of human brain. With this approach, the graph that defines the network can be dividen into 3 layers: input, hidden and output (see Figure 1). While the hidden layer can change the structure adopted, the input and output layer remain stable all the time, as the input and outputs of the system must remain the same.

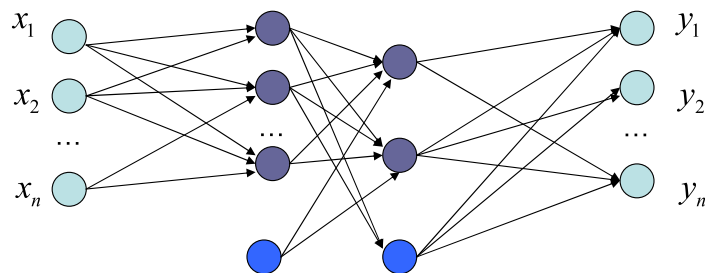


Figure 1: Structure of a neural network of  $n$  inputs,  $n$  outputs and 2 hidden layers.

The objective of a neural network system is to give an output due some input signals. Before the training of the neural network, ths system is initialized to its defaults values, and all the outputs (possible answers of the system) have the same probability. While the network is trained, the weights that define the connection between notes modified the

value, and depending on the input and hidden values, the structure can be also changed. That implies that it is possible to optimize the neural networks modifying the structure of the solution and modifying the way that the weights are calculated. During this project, two different approach have been used and compared on the calculation of the weights: back-propagation and genetic algorithm.

### 1.1 Genetic algorithm (GA)

The genetic algorithm is a search technique based on the concept of evolution[1], and in particular with the concept of the survival of the fittest[2]. The application of genetic algorithm on neural network make an hybrid neural network where the weights of the neural network are calculated using genetic algorithm approach. From all the search spaces of all the possible weights, the genetic algorithm will generate new points of the possible solution.

The first step to calculate its values, is to define the solution domain with a genetic representation (problem encoding) and a fitness function to determine the better solutions. Those two component of a genetic algorithm are the only problem dependent of the genetic algorithm approach. Once an initial population of elements (chromosomes or genotype[3]) have been created, the techniques used by this algorithms to converge to a solution of the problem are related to the evolutionary theory:

1. Selection → some chromosomes of the current population are selected to breed a new generation. A small number are selected randomly while the others are selected depending on how they fit better with a fitness function.
2. Genetic operations → once the first chromosomes have been defined by those that fits better the fitness function, the rest of the population is going to be created using genetic operations. The fitnesses of those new chromosomes will also be checked and compared with the worst chromosomes of the last generation in order to decide who will stay into the population.
  - (a) Mutation → modifying one or more bits (gens) of some chromosomes of the population.
  - (b) Crossover → crossing two or more chromosomes of the population between them.

A pseudo-code for this algorithm is:

1. Creation of the initial population.
2. while (!solution)
  - (a) Evaluate the fitness of all the chromosomes of the population.
  - (b) The best chromosomes will be selected to reproduce, using mutation and crossover.
  - (c) Substitute the worsts chromosomes of the previous generation by the new produced chromosomes.

This process was refered to as Simple Genetic Algorithm[4].

Finally, the fittest chromosome will be selected as a solution.

### 1.2 Description of the problem

The problem used to test the performance of the two methods is a Balance Scale, where the objective is to classify the inclination of a balance depending on the inputs. In this problem,

the data set has 4 attributes (input of the system) and the classification of the data (desired output of the system). From this data, it is possible to define the inputs and outputs of the system as:

Input:

1. Left weight.
2. Left distance.
3. Right weight.
4. Right distance.

Output:

1. Left balanced.
2. Balanced.
3. Right balanced.

Finally the data given was divided in two sets. Then, on the first attempt, the first set will be used as training data for the two approximations, and the performands will be tested. After that, the first set was used as test data instead of training data and the same process was made.

## 2 Implementation

Recalling the objectives of the project, it is necessary to find the weights that determine the connexion between the nodes. Two methods are going to be studied: back-propagation and genetic algorithm approach, with the same network topology. This topology is determined with the nature of the solution, as the relation between the input and the output of the problem is non-linear. That implies that, to be able to solve this problem, at least one hidden layer must be used.

The topology finally chosen was one hidden layer with 3 nodes, that with the specifications of the problem makes a neural network of 4 inputs, 1 hidden layer of 3 nodes and 3 outputs (see Figure 2). To solve this problem, a bias node must be used too in the hidden layer.

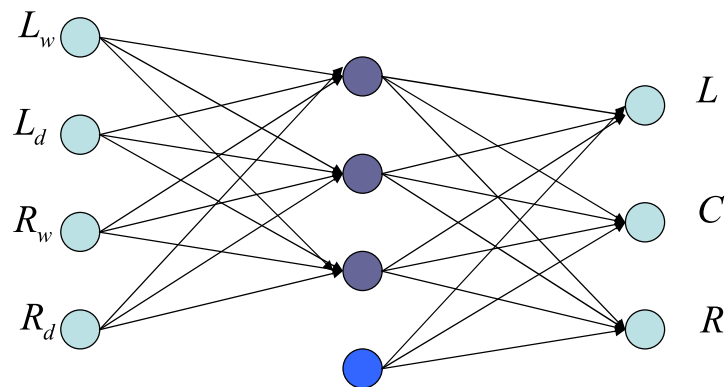


Figure 2: Structure of a neural network of 4 inputs, 1 hidden layer of 3 nodes and 3 outputs with 0 hidden layers.

## 2.1 Back-propagation implementation

The training of the network using the back-propagation implementation is directly related to the function cost chosen. Defining the squared error cost function.

$$J(w) = \frac{1}{2} \sum_i \left( y^{(i)} - out(x^{(i)}) \right)^2 \quad (1)$$

The next value for each weight is calculated from the gradient of the cost function.

$$w := w - \alpha \frac{\delta J(w)}{\delta w} \quad (2)$$

As it is indicated in its name, back-propagation implies that the first weights to be calculated in each iteration are the weights between the hidden layer and the output layer. After this first calculation, the weights of the previous level are calculated and so on, until finding the new values of the weights between the input and the first hidden layer.

## 2.2 Genetic algorithm implementation

Recalling the necessities of a genetic algorithm, a fitness function and a representation model are required. The solution with the genetic algorithm approach is going to use Equation (1) as the fitness function. To represent a chromosome, a string of bits will be used, and the crossover operation will be made only with half of the chromosome (see Figure 3).

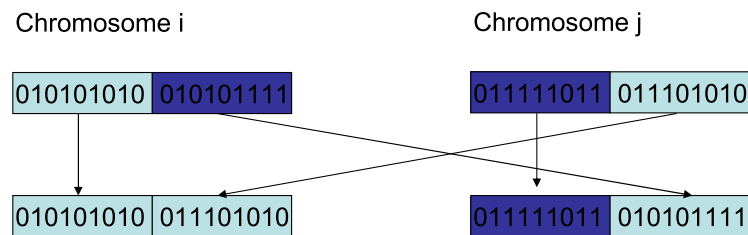


Figure 3: Crossover operation.

It is also important how the chromosomes are represented and how we select or create new chromosomes from the fittest elements. For each weight it is necessary to create some generations of chromosomes to find a good candidate, and in each generation it is necessary to have  $n$  chromosomes to select the fittest. In that problem,  $n = 50$  chromosomes per weight and generation were chosen to try to find the solution, that makes a total number of 600 chromosomes.

For simplicity, the cost function is calculated only for each node that we are working with, without crossing results, that could achieve a very high computational cost. Then, the algorithm to all the weights on each iteration will be:

```
Creation of the initial population of 12x50
chromosomes.
for (all_training_data)
    for (all_weights)
```

```

for (i = 1 : 50)
- Evaluate the fitness of all the chromosomes of
  the population.
- The best chromosomes will be selected to
  reproduce, using mutation and crossover.
- With the new chromosomes created from the
  fittest of the previous generation, a new
  generation is created.
end for
Evaluate the fitness for all the chromosomes of
the population.
Select the fittest chromosome of the population
as the new weight.
end for
end for

```

### 3 Simulation. Results

The comparison between the two methods gives interesting results. As expected, the GA approach gives better results than the back-propagation method with almost all the iterations used (using 6% and 50% as mutation and crossover probabilities). Only when the number of iterations is very small, the back-propagation method gives better precision performance.

In time consumption for the two algorithms, the result was not so positive for the genetic algorithm, although the time performance was still better than using back-propagation (see Table 1).

Table 1: Back-propagation vs Genetic Algorithms

	<b>Back-propagation</b>	<b>Genetic algorithm</b>
Time	86	72
Hits train data	36	46
Hits test data	32	37

### 4 Conclusions

The results of this project gives a good approximation of the benefits of a neural network system using genetic algorithms in front of back-propagation method, being superior this first method in precession and time consumption. But there are more variables to consider; in this problem, the values of the weights were "more or less" known, so one of the problems related with the genetic algorithm approach was solved. What would have happened if the margin of the weights was not known? The search space for the genetic algorithm would be much larger that the one that we were supposed to search, and the solution not only will converge slowly, but also with less precision.

Also, the back-propagation method uses a static value for  $\alpha$  (see Equation 2) that makes the weights converge in a non optimal way to the expected value. A proper calculation of  $\alpha$  each iteration will increase the performance of the method, although it will also increase the computational time.

It seems that using the genetic algorithm with neural network can make the neural network improve the results, although it is not necessarily sure for all the network topologies (or, at least, can not be deduced from the results obtained). It could be also interesting to include other paradigms to the genetic algorithms, as the Baldwin effect [], that could accelerate the learning of the algorithm increasing the average fitness of the individuals over a number of generations.

In future work, it can be also interesting a prior calculation to determine the topology of the best (calculated) network[], and related to the weight calculation, calculate initial values of the weights using GA and iterate using back-propagation. Those are problems of the neural network that can be solved using genetic algorithms.

Another fields where evolutionary algorithms can be applied outside the physical structure of the network are:

1. If the method selected is back-propagation, and a non-constant  $\alpha$  is chosen to solve the problem, finding the  $\alpha$  is a search problem that can be solved with GA.
2. Inside the programming of the genetic algorithm, to determine the margin of values of the weights.
3. Reduce the space of the training data to some data that defines the output with fidelity.

## References

- [1] Darwin, Ch. (1859) The origin of species by means of natural selection.
- [2] Whitley, D. (1994) A Genetic Algorithm Tutorial. Colorado State University, Computer Science Department.
- [3] Holland, J. (1975) Adaptation in Natural and Artificial Systems. University of Michigan Press.
- [4] Goldberg, D. (1987) Simple Genetic Algorithms and the Minimal, Deceptive Problem. In L.Davis, ed., Pitman *Genetic Algorithms And Simulated Annealing*.
- [] Baldwin, J.M. (1896) A New Factor in Evolution. *American Naturalist*, vol. 30, pp. 441-451.
- [] Fiszlelew, A., Britos, P., Ochoa, A., Merlino, H., Fernandez, E., Garca-Martnez, R.(2007) Finding Optimal Neural Network Architecture Using Genetic Algorithms. *Research in Computing Science* 27, pp. 15-24.
- [] Aho, I. & Kemppainen, H. (1997) Searching Neural Network Structures with L Systems and Genetic Algorithms. *Series of publications A. A-1997-15*. University of Tampere.
- [1] Alexander, J.A. & Mozer, M.C. (1995) Template-based algorithms for connectionist rule extraction. In G. Tesauro, D. S. Touretzky and T.K. Leen (eds.), *Advances in Neural Information Processing Systems* 7, pp. 609-616. Cambridge, MA: MIT Press.
- [2] Bower, J.M. & Beeman, D. (1995) *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural Simulation System*. New York: TELOS/Springer-Verlag.
- [3] Hasselmo, M.E., Schnell, E. & Barkai, E. (1995) Dynamics of learning and recall at excitatory recurrent synapses and cholinergic modulation in rat hippocampal region CA3. *Journal of Neuroscience* 15(7):5249-5262.