Alex Ksikes
ak107@cornell.edu
05/2002

Evolutionary Artificial Neural Networks

In this project we address the problem of learning an ANN through the use of a genetic algorithm. Such a method has 2 main advantages over standard back propagation. The genetic algorithm globally searches for a solution, therefore minimizing the risk of getting stuck into a local minimum. The genetic algorithm could be used to explore different topologies.

Problem definition:

There are two parts to the project. The first part uses a population of ANN with a fixed number of hidden neurons. The second part the number of hidden neurons varies to explore different topologies. In both parts we consider only one hidden layer feed forward ANNs.

Part 1: Number of Hidden Neurons Fixed

1) The Learning System

The Target Function:

We will be testing the results on the character recognition problem seen in class for the ANN assignment. We chose this problem because of its simplicity. It offered an easy way to test the genetic algorithm.

Representation:

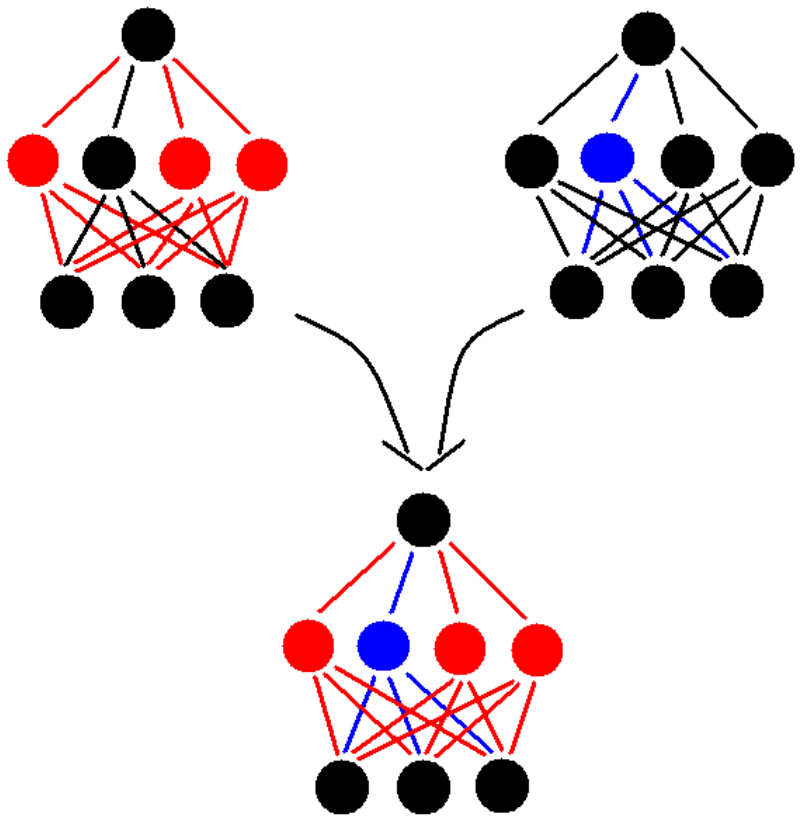
A chromosome is an array of the weights of each hidden neuron. Each entry in this array has 2 entries. The first one represents the weights going into the neuron, the second one the weights going out of the neuron. We chose such a representation in order to preserve the neurons through crossing over. The idea is that an ANN was ranked highly because it possesses some neurons that have learned part of the problem. This schema is to be opposed with a simpler one where a chromosome is a concatenation of weights. We did not choose this latter because we believe the ANN performs well not because of its weights (taken into some order) but rather because of its neurons.

System Structure:

We chose as the fitness function the RMS of the corresponding ANN when evaluated over the training set. A uniform cross over was used to generate the offspring. The

mutation operator consisted of adding weights normally distributed between 1 and –1 to the every edge.

Crossing Over Between 2 ANN



The Learning Algorithm:

The genetic algorithm generates a population with weights uniformly chosen between –1 and 1. A rank selection is used to select the best hypotheses. The algorithm ends after a chosen number of iterations.

2) Experiments

The Data Sets:

We used the character recognition data sets. The training set was used to compute the fitness of each hypothesis. The evaluation set was used to have an idea of the generalization power of the best hypothesis.

Learning:

We performed 150 iterations (150 generations) with a population size of 50 hypotheses with 7 hidden neurons each. The crossover rate was set to 0.6 and the mutation rate was set to 0.15.

Evaluation/Tests:

The system performance was evaluated using the evaluation set.

3) Results

Refer to graph 1 to 4.

4) Data analysis

We plotted the training and evaluation accuracy of the best solution. The results are surprisingly good. We achieved for the last iteration an evaluation accuracy of 0.72 (graph 1).

We compared these results with the one obtained by backpropagation in stochastic mode with a learning rate of 0.1 and momentum of 0.01. Since one iteration of the GA corresponds to 50 network evaluations (one for each hypothesis) and back propagation performs 2 network evaluations per iteration (the same feed forward stage performed by the GA fitness function, followed by the adjustment of the weights) we recorded the value of the training accuracy of backpropagation after every 25 iterations (one iteration of GA corresponds to 25 iterations of back propagation) (graph 2).

Over the training set backpropagation converges faster towards a solution than the GA. However backpropagation over fits the data quite strongly. The best observed evaluation accuracy of the backpropagation algorithm is 0.732 at iteration 20. On the other hand no over fitting is observed for the GA. It ended with an evaluation accuracy of 0.726 at iteration 150. Therefore in the long run the GA algorithm could very well outperform backpropagation.

We perform more tests with different parameters (graph 3 and 4) to prove that over fitting must generally happen later throughout the execution of the genetic algorithm. However due to limitations in the computing environment we have been forced to keep the maximum number of iterations to a minimum.

Part 2: Number of Hidden Neurons Unfixed

1) The Learning System

The Target Function:

We use the same target function as for part 1.

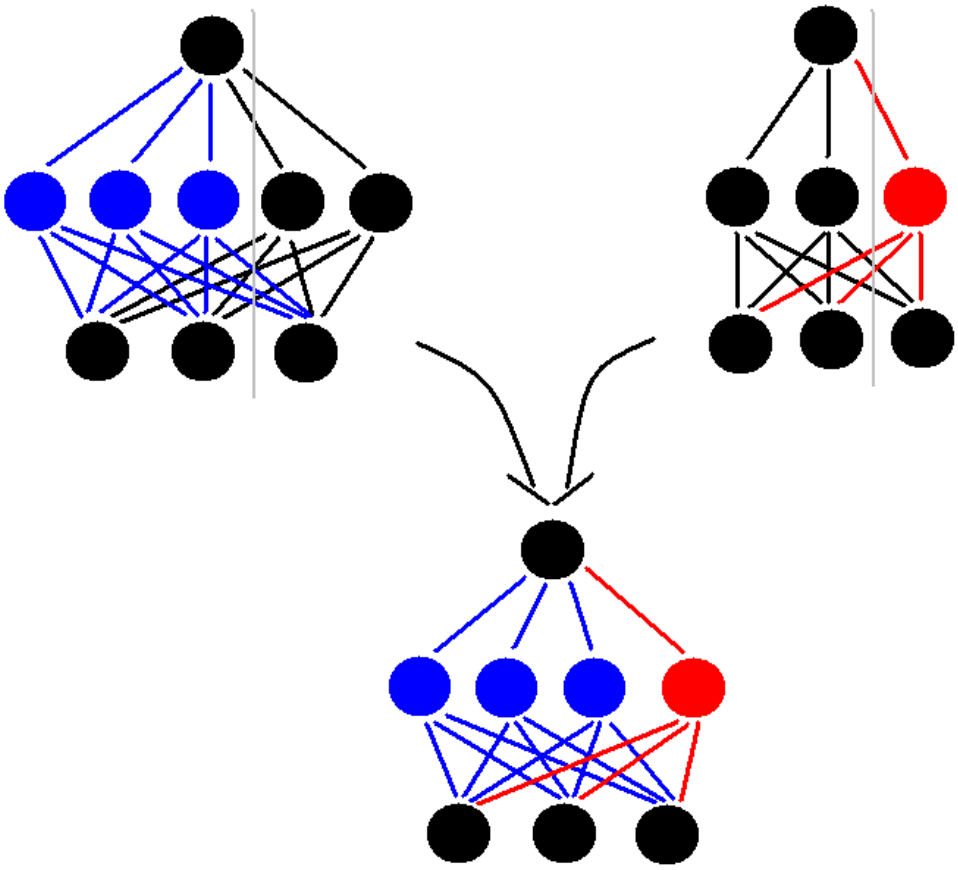
Representation:

The same representation as in part 1 is used.

System Structure:

The topology was explored by using a standard cross over operator. The offspring is generated by merging the left part of the first parent with the right part of the second parent. However this method tended to generate huge hypotheses that will poorly generalize over the validation set. To tackle this problem we introduced a fitness function to be the RMS rounded up to 3 significant digits. When two hypotheses have the same fitness, the smaller one is favored over the larger one. As for part 1 the mutation operator consisted of adding weights normally distributed between 1 and –1 to the every neuron. We did not choose to randomly delete neurons or add neurons as such a method would destroy too much of the already learned network.

Crossing Over Between 2 ANN of Different Sizes



The Learning Algorithm:

The initial population consisted of hypotheses with a randomly generated number of hidden neurons (between 2 and some chosen value). As for part 1 the algorithm assigns weights uniformly distributed between –1 and 1 for each hypothesis. A rank selection is used to select the best hypotheses but this time if two hypotheses have the same fitness (rounded RMS) then the smaller one is favored. The algorithm ends after a chosen number of iterations.

2) Experiments

The Data Sets:

We used the character recognition data sets. The training set was used to compute the fitness of each hypothesis. The evaluation set was used to have idea of the generalization power of the best hypothesis.

Learning:

We performed 100 iterations with a population size of 13 hypotheses. The initial size of each hypothesis was chosen randomly between 2 and 5 hidden neurons. The crossover rate was set to 0.6 and the mutation rate was set to 0.15.

Evaluation/Tests:

The system performance was evaluated using the evaluation set. The number of hidden neurons was also taken into account.

3) Results

Refer to graph 5 and 6.

4) Data analysis

We achieved a training accuracy of 0.67 and an evaluation accuracy of 0.65. The best hypothesis to achieve such accuracy had 8 hidden neurons. We also note, as expected, that smaller hypotheses perform generally better over the evaluation set than do larger ones. Our method of favoring smaller hypotheses did pay off but it should be kept in mind that in the long run the algorithm would still end up producing very large hypotheses that will poorly generalize over the evaluation data. Therefore further improvements must be brought to the algorithm in order to keep the size of each hypothesis as small as possible.

Summary

We presented two methods of learning an ANN with a genetic algorithm. In both methods we assumed one hidden layer feed forward ANNs. The first method kept the number of hidden neurons of each hypothesis fixed. The results were surprisingly good

and let us believe that it could outperform backpropagation (at least for this problem). The second method did not require the number of neurons to be fixed. The results were not as conclusive as for the first method.

Future Work

The first part could be generalized to multi layer ANNs. A chromosome would represent the neurons in each hidden layer. The crossover operator should be performed between neurons in the same layer.

Another interesting approach would be to use a combination of backpropagation and GA to seek for the best hypothesis. We could combine the speed of backpropagation to reach a local minimum with the generalization power of a genetic algorithm.