Evolving Neural Network Structures by Means of Genetic Programming

Wolfgang Golubski and Thomas Feuring

Universität–GH Siegen, FB 12: Elektrotechnik & Informatik
Hölderlinstraße 3, D–57068 Siegen, Germany
{golubski, feuring}@informatik.uni-siegen.de

Abstract. The goal of this paper is to present a more efficient way to automatically construct appropriate neural network topologies as well as their initial weight settings. Our approach combines evolutionary algorithms and genetic programming techniques and is based on a new network encoding schema where instead of a string like encoding the graph representation of neural nets is used. This way of "encoding" reduces the computational expense and leads to a greater variety of network topologies.

1 Introduction

Neural nets are known to be a powerful tool for a wide range of problems, such as classification, recognition and others. Their advantage is that they can be trained by using a set of input/output pairs. The underlying input/output relationship can be approximated by a suitable neural net during the training process.

It is well known that feedforward nets are universal approximators which means that a three layered feedforward net can approximate any continuous function on a compact domain. Since there are numerous of different network topologies and even more different weight initializations finding a suitable network topology is a quite complex optimization problem. But how can we find a suitable network topology with appropriate weight values? Currently, neural nets are essentially constructed by means of (1) backpropagation algorithms [7], (2) genetic algorithms [1,3], or (3) genetic programming strategies [2]. Each approach has its own well known advantages but also some important disadvantages like the followings:

– *Backpropagation:* Many different network topologies have to be trained and tested before a suitable net is obtained. The training process highly depends on the initialized network weights. Each network topology has to be trained several time before it can be accepted. This procedure is obviously very time consuming so only a few network topologies can be trained and tested. Some experience is needed in order to find a "good" net. A second problem which occurs is that by permutation of the weights an identical net can be obtained. This results in an error surface having many global minima so that it usually appears rough in many dimensions. This can slow down Backpropagation.

R. Poli et al. (Eds.): EuroGP'99, LNCS 1598, pp. 211–220, 1999.

- *Genetic Algorithm:* Neural nets have to be coded into a string-like representation (of binary or real values) so that crossover and mutation operators can be used to optimize the net topology [10]. Directed search on the set of all weights of a net can be used for the task of finding a suitable network topology. So the initial net structure remains (nearly) unchanged.
- *Genetic Programming:* The coding process here (e.g. nets are encoded in grammar trees [4,5], or in Lisp functions [8]) is the most difficult part because of its computationally inefficiency. The networks to be encoded are different in their topology as well as in their number of weights. Again symmetry of the net weights can slow down the search for "good" nets.

To overcome these facilities we propose an approach [6] which combines the ideas of genetic algorithms and genetic programming. The net topology is no longer coded into a string representation but the graph of the net is used as a representation itself. Crossover operators as well as mutation operators can be defined on these graphs instead on strings or trees. Some advantages of this representation are that the process of evolving a suitable net is speed up since no coding and decoding is required and that the permutation problem can be reduced. Furthermore this approach can be seen as a generalization of other evolutionary based neural net creating approaches since we are using a more general coding scheme.

The main contributions of our paper are:

- A simple representation of neural networks by directed, weighted graphs where nodes represent neurons and the incoming arcs of the inner nodes are attached with real values called weights;
- Deriving recombination of neural nets by randomly selecting and exchanging subgraphs between neural nets;
- Defining various crossover operators;
- Presenting experimental results delivered by benchmarks on three test functions.

The paper is organized as follows. After this introduction we describe the basic structure of the neural nets to be examined in this paper. In Section 3 we focus on the basic structure of the genetic programming based algorithm used for evolving the neural nets. In that section we introduce different crossover and mutation operators. In the following section we present a detailed analysis of the tests performed and results obtained with our approach. Finally, some conclusions as well as some proposals for future works are given in Section 5.

2 Neural Nets

The neural nets we are interested in are feedforward neural nets consisting of an input layer, one or more hidden layers as well as an output layer, see Figure 1. At this stage of the project we are using exactly two or three hidden layers. Furthermore, all neurons of a layer are fully interconnected with all the neurons of the next layer.

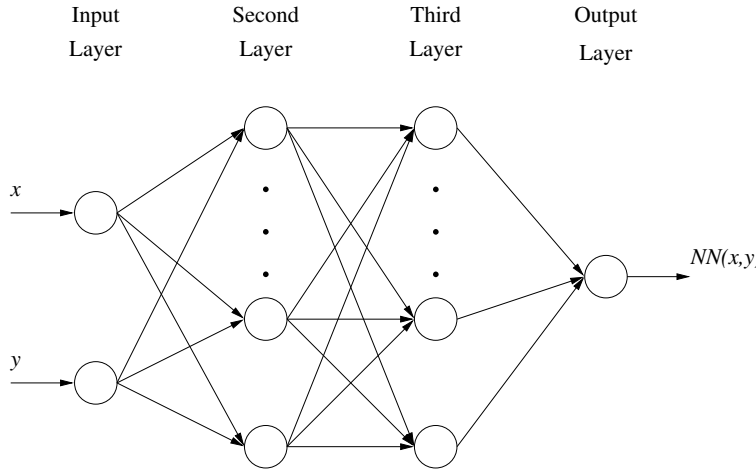

Fig. 1. Three Layered Neural Net.

The input neurons simply distribute the input data to all the neurons of the second layer. The later neurons compute the weighted sum of their inputs. This result is mapped by a transfer function to the output of the corresponding neuron. In ours tests we are using the sigmoidal function $s(x) = (1+\exp(-x))^{-1}$ to compute the output of the neurons. These outputs are sent via the directed arcs to the neurons of the third layer. These neurons act in much the same way as the neurons of the second layer. Finally, the output neurons compute the weighted sum of the outputs of the third layer neurons. Here the transfer function can be the identity function $i(x) = x$ if the desired output is not normed to the interval $[0, 1]$ or it can be a sigmoidal function as in the neurons of previous layers if the desired output is normed to $[0, 1]$.

3 Our Approach

Koza [9] used the idea of Genetic Algorithm [1] to evolve LISP programs. Hereby starting with a set of LISP programs new LISP programs are evolved by recombination until a sufficient good solution has been found. Programs are represented by their syntactic tree structure. Crossover operators operate on the tree structures by randomly selecting and exchanging subtrees between individuals which are selected according to a fitness function. This way a new tree representing a new program is generated. In contrast to evolutionary algorithms no mutation operators are used in the genetic programming approach.

In our method we combine the genetic algorithm concept with the genetic programming ideas by using graphs instead of trees. Since feedforward nets as shown in Figure 1 and 2 look like directed weighted graphs this structure is obviously a suitable candidate for representing neural nets. This way we reduce the computational expense needed for coding and decoding of neural nets. In

the graph representation each arc between two nodes of the graph is associated with a real value representing the weight of the corresponding neural net. In our approach the graph no longer represents a certain program but a neural net. So, crossover operators as well as mutation operators can be defined on these graphs. This is why our approach can be seen as a combination of genetic programming and genetic algorithms.

The whole evolution process starts with the evaluation of the fitness of all individuals (neural nets). As an indicator for the fitness we used the mean squared error on a given training set consisting of input and output pairs, which describe the input output relation to be approximated by a neural net. Therefore, the smaller the error the better is the fitness of the corresponding net. Out of the fittest nets the (λ, μ) selection process, selects the best μ individuals out of the λ members of the generation.

The recombination process now builds a temporary generation by applying a crossover operator to the previously selected individuals (of the previous generation). In order to build the temporary generation we randomly choose 10% or 20% (reproduction rate) of the μ fittest neural nets to become members of the temporary generation. Since the population size of the temporary population is λ the remaining neural nets are constructed by recombination. So, first two "parents" are chosen randomly from the μ selected members. Then the crossover operator first chooses the hidden layer which is recombined in the exchange process. Now two subgraphs (consisting of nodes and the arcs to these nodes) of the chosen layer of both parents are randomly taken and exchanged in order to produce two temporary individuals. Both are taken into the temporary generation. The process of choosing parent nets is repeated until the temporary population consists of λ individuals. Two different crossover operators are used in our experiments. The first one is non-restrictive which means that the subgraph to be exchanged can consist of a randomly chosen number of nodes according to the number of nodes of the corresponding layer. Figure 2 describes this situation. In this Figure the second layer is chosen for the recombination process. A three node subgraph (Parent A) and a one-node subgraph (Parent B) are selected in the second layers and exchanged to produce Child 1 and Child 2. For Child 1 the three-node subgraph is substituted by a one-node subgraph. Since the number of input neurons is identical we only have to focus on how to change the arcs from the substituted subgraph to the nodes of the third layer. Since the third layer of Parent A only has three nodes but there are four arc in the chosen one-node subgraph of Parent B we randomly delete one arc and use the remaining three to connect to one-node subgraph with the three nodes of Parent A to obtain Child 1. For Child 2 we proceed similarly except that for each of the nodes of the three-node subgraph a new arc (and a weight) is randomly generated. Finally, both children are taken into the temporary generation. We call this a non-restrictive crossover operator since the graph can grow rapidly. Since we are interested in small nets, we have tested two versions of the crossover strategies: one where the randomly chosen part is only restricted by the number of neurons in its layer (the non restrictive) and one where the randomly chosen part cannot

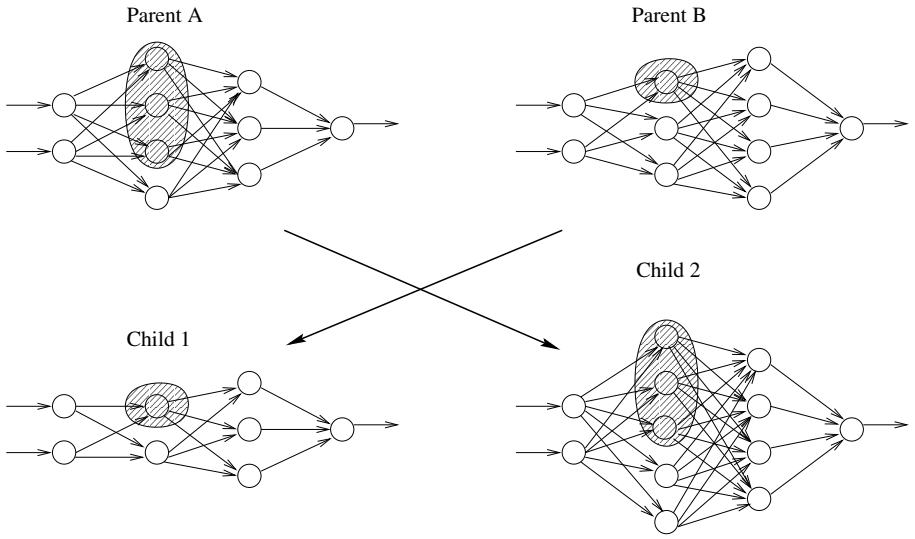

Fig. 2. Crossover Operator (non restrictive).

be greater than two neurons (the restrictive version). For this crossover operator only one-node or two-node subgraphs can be chosen for the parent graphs (see Figure 3). This way the growth of the nets is reduced. Therefore, this crossover operator is denoted as restrictive. The most important parameter adjustments are summarized in Table 1.

Table 1. Parameter Adjustment

Parameter	**Version 1**	**Version 2**
Population size	1000	1000
max. generation	200	200
Recombination rate	80%	90%
Reproduction rate	20%	10%
Number of fittest nets	72	72
Fitness type	MSE	MSE
Fitness threshold	0.01	0.01

After recombination and reproduction step a mutation process follows. According to a predefined mutation rate the mutation process builds a new member of the new generation by adding to each weight a value $\exp(\tau * N(0,1))$ where $N(0,1)$ stands for a normally distributed random variable having expectation value zero and standard deviation one and τ is an additional parameter which is set to $n^{-0.5}$, where n is the number of network weights needed for the corresponding net (see [1]).

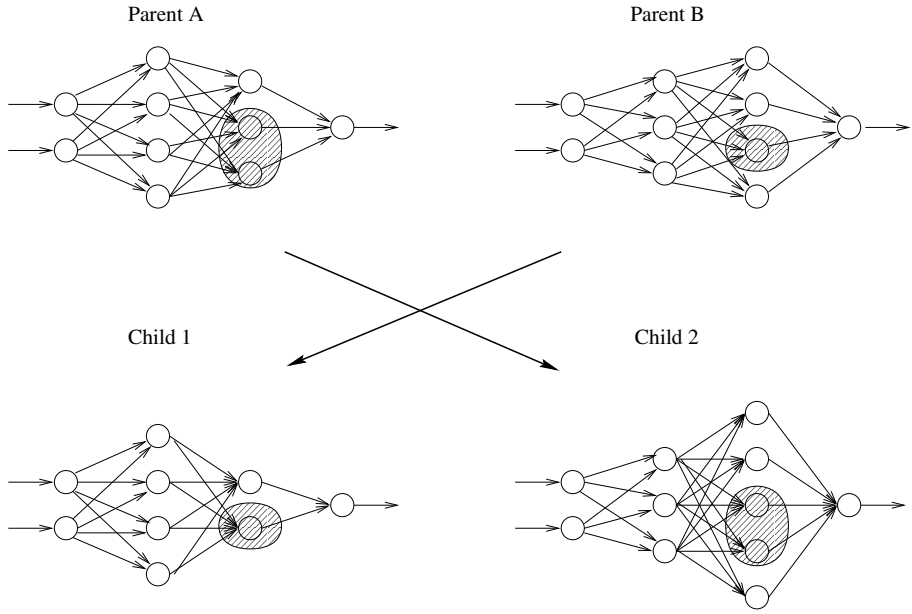

Fig. 3. Crossover Operator (restrictive).

Now again the selection operator finds the best members of the population, which are used by the recombination operator to produce a temporary generation and the mutation operator which mutates each temporary element in order to construct the next generation. This evolution process of the neural net structures continues until a given number of generations is reached, or a given fitness level is obtained.

This concept was implemented and tested using different benchmark data sets as well as different settings. All results obtained by this approach are discussed in detail in the following section.

4 Results

The application of our implemented algorithm is currently going on. We present results of three test functions (sin, Rosenbrock, Schaffer (see [10])) which were trained 5 times using two (NN2L, rNN2L) and three (NN3L, rNN3L) hidden layers and crossover operator without (NN2L, NN3L) and with limited (restrictive) (rNN2L, rNN3L) growth, altogether in two parameter versions (v1 and v2). E.g. rNN3Lv2 names the neural net evolution having three hidden layers, using the restrictive crossover operator based on the parameter settings of version 2.

4.1 Successful Runs

A test run is classified as successful if within 200 evolution steps the mean squared error is less than 0.01. The number of successful runs versus unsuccessful runs are given in Table 2 and 3. In the most cases the 2 hidden layer nets show better results than the 3 hidden layer nets but the schaffer example indicates that sometimes a 3 hidden layer net leads to a better net.

Comparing unrestricted, growing, crossover (NN2L, NN3L) and its restricted version (rNN2L, rNN3L) we surprisingly observe that the restricted crossovers show significantly better results than its counterpart.

Table 2. Successful Runs for the Different Crossover Operators and Net Sizes for the Parameter Version 1

	NN2Lv1	NN3Lv1	rNN2Lv1	rNN3Lv1
sin	2:3	3:2	4:1	2:3
rosenbrock	2:3	0:5	4:1	0:5
schaffer	1:4	1:4	0:5	3:2

Table 3. Successful Runs for the Different Crossover Operators and Net Sizes for the Parameter Version 2

	NN2Lv2	NN3Lv2	rNN2Lv2	rNN3Lv2
sin	2:3	0:5	5:0	3:2
rosenbrock	3:2	3:2	5:0	3:2
schaffer	4:1	5:0	1:4	4:1

4.2 The Sizes of the Best Nets

Considering the topological size, see Table 4 and 5, of the evaluated nets the restrictive crossover is the more appropriate one. The non restrictive version leads to an unlimited growth of the net sizes, see the results of NN3L. But again like above the schaffer-function is a good example that greater net structures are thoroughly rich in meaning.

4.3 The Number of Generation Steps Needed

Similar to the results above, see Table 4 and 5, we obtain that the restrictive version is superior to the non-restrictive one according to the number of generation steps of the evolution process needed to get a succesful run. E.g. rNN2Lv2 and NN2Lv2 applied to sin shows that 4 of 5 test runs of rNN2Lv2 finish more faster than the best NN2Lv2 run does.

Table 4. Results about the Net Sizes (Number of Neurons in the Hidden Layers) of the Best Successful Nets and the Number of Generation Steps needed

	NN2Lv1	NN3Lv1	rNN2Lv1	rNN3Lv1
sin	10-2 [187]	14-2-9 [173]	12-2 [128]	8-2-2 [88]
	18-2 [121]	23-2-25 [158]	8-9 [106]	15-10-5 [138]
		39-2-25 [177]	16-4 [162]	
			13-8 [112]	
rosenbrock	11-6 [175]		7-2 [145]	
	30-2 [94]		12-3 [94]	
			12-9 [112]	
			15-8 [167]	
schaffer	23-2 [187]	17-2-2 [83]		10-2-2 [133]
				9-5-2 [107]
				13-8-2 [139]

Table 5. Results about the Net Sizes (Number of Neurons in the Hidden Layers) of the Best Successful Nets and the Number of Generation Steps needed

	NN2Lv2	NN3Lv2	rNN2Lv2	rNN3Lv2
sin	52-2 [187]		14-8 [142]	14-13-7 [180]
	60-2 [155]		9-19 [96]	9-6-10 [141]
			15-6 [166]	12-7-11 [181]
			18-10 [147]	
			19-8 [151]	
rosenbrock	30-2 [165]	25-5-2 [102]	13-13 [152]	8-9-16 [156]
	45-12 [155]	12-25-2 [89]	17-14 [128]	10-9-4 [148]
	73-3 [112]	38-2-13 [193]	20-11 [90]	13-13-8 [130]
			10-2 [80]	
			15-13 [132]	
schaffer	11-2 [167]	8-2-2 [67]	14-5 [152]	8-3-2 [125]
	9-2 [90]	8-2-2 [132]		8-6-5 [57]
	11-2 [138]	16-7-2 [169]		11-2-2 [89]
	27-2 [118]	14-2-2 [116]		11-6-2 [134]
		14-7-2 [67]		

4.4 Parameter Version 1 vs Parameter Version 2

Here the unique winning adjustment is version 2 with a higher recombination rate (90%). All runs of the test functions except NN3Lv2 applied to sin show better results with respect to successful runs and the number of generation steps needed. A comparison of the net sizes shows a complex picture. Sometimes the algorithm using parameter version 1 delivers smaller nets and sometimes the parameter version 2 algorithm evaluates smaller nets than its competitive parameter version.

5 Considerations about the Computational Expense

In this section we want to compare the computational expense of training a layered neural net with backpropagation and obtaining a neural net with out genetic programming based approach. The following considerations are very roughly since they will ignore the fact that network sizes occurring in our genetic programming based approach are different. However this assumption will simplify the following discussion.

In both approaches the same computational expense is needed for evaluating the error of a given network since in both nets a feedforward pass is performed. The weight updates using the backpropagation algorithm is more expensive than the corresponding forward pass. Since the recombination and mutation processes are also more time consuming than a forward pass (two passes through each individual have to be performed) we estimate both update methods to be nearly the same (from a computational expense point of view).

If we further assume that 1000 elements are evaluated in the genetic programming based approach we end up with the following comparison. One generation in our approach is roughly as expensive as 1000 backpropagation steps for a given network with a given weight initialization. So, if we would net 200 generations for finding a suitable net topology ending up with an error of 0.001, we could need 200.000 backpropagation steps in order to find a suitable net. Of course, using the backpropagation algorithm would lead to testing different networks with different weight initializations. Depending on how lucky the user is he could find a good net with a smaller error than 0.001 within 200.000 steps. Since some experiences are needed to find a good net it might also take some longer to find a good net. The point here is that using our approach there would be no need for a user to "search" for a suitable net. So, with our approach this human resource is saved.

Let us mention one final point. Our approach only leads to a got initialization of the most suitable network. The resulting net can easily be trained by a second order descent methods like the quickprop algorithm. This method leads to a faster descent as the backpropagation. Since the quickprop only guarantees local convergence (the algorithm converges only if the error is sufficiently close to the minimum) it is not suitable (from a theoretical point of view) to use this algorithm instead of the backpropagation.

6 Conclusions and Further Works

We presented a genetic programming based neural net topology initialization approach and showed results on three test functions applied to different crossover strategies and two genetic parameter adjustments. The tests are very promising even if the sizes of the nets obtained by the program could be smaller. The restrictive crossovers show better results than the unlimited growing crossovers and a higher recombination rate is much better than a smaller one.

Currently we are working on (1) an extension of the exchanging subgraphs property to yield a crossover operator where subgraphs consisting of neurons of two succeeding layers are participated; (2) a direct comparison of backpropagation strategies and our approach and (3) a hybrid extension of our algorithm where after a fixed number of population steps a fixed number of backpropagation steps should help to improve the overall quality of the evolved nets.

References

1. Bäck, T.: "Evolutionary Algorithms in Theory and Practice : Evolutionary Strategies, Evolutionary Programming, Genetic Algorithms". Oxford University Press, New York, 1996.
2. Banzhaf, W., Nordin, P., Keller,R.E., Francone, F.D.: "Genetic Programming: An Introduction". Morgan Kaufmann Publishers, 1998.
3. Fogel, D.B.: "Evolutionary Computation: Toward a New Philosophy of Machine Learning". IEEE Press, Piscataway, 1995.
4. Gruau, F.: "Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm". PhD-Thesis, University of Grenoble, France, 1994.
5. Gruau,F., Whitley, D., Pyeatt, L.: "A Comparison between Cellular Encoding and Direct Encoding for Genetic Neural Networks", Proc. of Genetic Programming 1996, Stanford University, pp. 81–89.
6. Golubski, W., Feuring, T.: "Genetic Algorithm based Neural Network Initialization: Part 1", to appear: International Conference on Computational Intelligence for Modelling Control and Automation 1999, Vienna, Feb. 1999.
7. Haykin, S.: "Neural Networks - A Comprehensive Foundation". Macmillan College Publishing Company, New York, 1994.
8. Koza, J.R., Rice, J.P.: "Genetic generation of both the weights and architecture for a neural network", In Proceedings of International Joint Conference on Neural Networks, Seattle, Los Alamitos, CA: IEEE Press, Volume II, pp. 397-404, 1991.
9. Koza, J.R.: "Genetic Programming II". Cambridge/MA: MIT Press, 1994.
10. Whitley, D.: "Genetic Algorithms and Neural Networks". in: Periaux, J. and Winter, G. (eds) "Genetic Algorithms in Engineering and Computer Science", 1995.