Genetic Weight Optimization of a Feedforward Neural Network Controller

Dirk Thierens*, Johan Suykens, Joos Vandewalle and Bart De Moor

Department of Electrical Engineering
ESAT-lab K.U.Leuven
Kardinaal Mercierlaan 94
B-3001 Leuven
Belgium

Abstract

The optimization of the weights of a feedforward neural network with a genetic algorithm is discussed. The search by the recombination operator is hampered by the existence of two functional equivalent symmetries in feedforward neural networks. To sidestep these representation redundancies we reorder the hidden neurons on the genotype before recombination according to a weight sign matching criterion, and flip the weight signs of a hidden neuron's connections whenever there are more inhibitory than excitatory incoming and outgoing links. As an example we optimize a feedforward neural network that implements a nonlinear optimal control law. The neural controller has to swing up the inverted pendulum from its lower equilibrium point to its upper equilibrium point and stabilize it there. Finding the weights of the network represents a nonlinear optimization problem which is solved by the genetic algorithm.

keywords: genetic algorithm, feedforward neural network, global optimization, nonlinear optimal control.

1 Introduction

Several authors have shown that single hidden layer feedforward neural networks (NNs) are universal approximators for any continuous mapping [1,2,3]. Unfortunately these theoretical proofs give very little guidance on how to obtain a good network for a specific problem. First there is the problem of network architecture: how many hidden layers do we need, how many neurons in each hidden layer, and what connectivity will give us optimal performance? Second there is the problem of weight determination: how do we get the connection weights once we have chosen a particular network topology? In this paper we are mainly concerned with the latter problem. Specifying the weights of a NN is mostly viewed as an optimization process where the goal of the computation is to find an optimal value of an error function. The most commonly used algorithms are backpropagation, conjugate gradient methods and variable metric methods. Although there are considerable differences between these algorithms they do have one significant property in common: they all are local optimization algorithms. Starting from an initial random value a sequence of neighboring points is generated by extracting local information of the search space. Depending on the particular algorithm used this information consists of the function values, the first- and/or second derivative. For non-trivial problems however the error surface is high-dimensional and contains many local optima. By using a local optimization algorithm on such a function there is a significant risk to convergence to some bad value, and in practice people simply do multiple runs with different random starting points. The amount of function evaluations for one single run is usually quite substantial so the ratio between the number of restarts and the number of local function evaluations is typically very low which in fact makes this approach a poor global optimization algorithm.

In this paper we discuss the use of a genetic algorithm (GA) to search the neural network weight space in a global way. In the next section we first look at the functional equivalent symmetries that exist in a wide class of NNs. Section 3 discusses the problems these symmetries cause for the GA, and some changes to the straightforward genotype representation are offered. Section 4 applies the ideas to the optimization of a neural network control function that has to swing up the inverted pendulum and stabilize it in the upper equilibrium point. Experimental results for optimizing the network with and without the proposed recombination modifications are compared.

*email: thierens@esat.kuleuven.ac.be

2 Functional Equivalent Symmetries in Feedforward Neural Networks

The functional mapping implemented by a single hidden layer feedforward network is not unique to one specific set of weights. The same mapping is also obtained by a number of different NNs. What characterizes these networks is that they all are a member of a finite group of symmetries defined by two transformations. Any member of this group can be constructed from any other member by a sequence of these transformations. The first transformation is defined at the single hidden node level. The second is defined at the hidden layer level.

2.1 Hidden Node Redundancy

The most frequently used class of feedforward neural networks consists of hidden nodes that sum their weighted inputs and subsequently apply a transfer function to produce their output value. A number of different transfer functions exist but most of them are odd. Examples are the linear threshold, the logistic and the hyperbolic tangent function. Since all these transfer functions are odd the output of the network does not change if we flip the sign of all the incoming and outgoing weights of a hidden node.

We can choose any combination of the n hidden neurons to flip their weight signs so there are $\sum_{i=0}^{n} \left(\begin{array}{c} n \\ i \end{array} \right) = 2^n$ structurally different but functionally identical networks generated by this transformation.

2.2 Hidden Layer Redundancy

A second functional equivalent group of networks is situated at the hidden node layer level. Suppose that we have a network with $h_1 h_2 \ldots h_n$ as hidden nodes. The mapping implemented by the network does not change if a particular hidden node with all its incoming and outgoing weights is exchanged with another neuron and its weights. For instance the networks $h_1 h_2 \ldots h_n$ and $h_2 h_1 \ldots h_n$ are equivalent, even though the first and second neuron have changed their position in the hidden layer. Obviously we can permute any of the n neurons so the total number of functional equivalent networks by this transformation is $n!$.

Since the two transformations are independent of each other, there is a total of $2^n n!$ functional equivalent but structurally different networks. Recently it has also been proven that at least in the case of a single hidden layer, one output neuron and a tangent hyperbolic transfer function the weights within this group of symmetries is unique [4], so there are exactly $2^n n!$ redundant networks for a specific mapping.

For the traditional local weight optimization algorithms this redundancy poses no problem since they only look at a small part of the search space. Global optimization algorithms however will try to explore the whole connection weight search space and this is a factor $2^n n!$ bigger than it really ought to be for the network to function as a universal approximator. For the genetic algorithm the problem is not only one of scale but also of crossover efficiency: functional equivalent near optimal networks often give rise to totally inappropriate networks after recombination because their weight structure is only equivalent up to a certain amount of function invariant transformations.

In the next section we first look at a straightforward genotype representation. The consequences of the functional equivalent groups are discussed and finally a method is offered to eliminate these redundancies.

3 Genotype Representation of Feedforward Neural Networks

3.1 Straightforward Genotype Representation

The standard approach in genetic algorithm practice is to represent the search space simply by concatenating all the parameters in a binary string - the genotype. Parameters close to each other on the genotype are more likely to be processed as a whole because it is less probable that a linkage biased crossover will disrupt them. Whenever a set of parameters form a functional unit it is therefore better to encode them tightly at the genotype. In neural networks the incoming and outgoing weights of a single hidden node form a high dimensional hyperplane, so we want to place them together on the string. The order with which the hidden neurons are placed on the genotype is irrelevant for the mapping performed by the network. Unfortunately for the crossover operator this is not the case: suppose we have a network with two hidden neurons $h_1 h_2$ and $h_1\prime h_2\prime$ with h_i and $h_i\prime$ representing similar hyperplanes. The genotype representation of the first network might be $h_1 h_2$ and for the second $h_2\prime h_1\prime$. When we recombine the two networks by crossing between the two nodes then one

offspring inherits h_1 and $h_1\prime$ and the other h_2 and $h_2\prime$. The new neurals nets will almost certainly have a very high error value. Ideally we want to exchange functional similar hidden neurons, and the following two paragraphs discuss a way to achieve this.

3.2 Hidden Layer Redundancy Elimination

The goal of the crossover operator in GAs is to take the partial solutions of two individuals and recombine them to form a better solution. In feedforward NNs with global defined transfer functions, the hidden nodes represent hyperplanes and we want to recombine the good hyperplanes of two NNs to create a better performing network. It is important however that the offspring inherits all the hidden nodes that are necessary to implement the desired mapping. To prevent that recombination would place the functional similar hidden neurons on the same offspring, we rearrange the hidden nodes before crossover is applied such that similar neurons are in the same position. In order to do this we need a way to easily identify the functionality of the hidden neurons and their connecting weights.

The approach we propose here is to look at the signs of the incoming and outgoing weights of every hidden node: the position of the hyperplane is predominantly determined by the weight signs. Hidden neurons that have most of their weight signs in common will be placed at the same position in the genotype before crossover is applied. The reordering of the neurons of one of the two recombining genotypes is done with a simple greedy algorithm. For convenience let us call parent1 the genotype that will be reordered to match the ordering of parent2. First we look for the hidden neuron in parent1 that best matches the first hidden neuron in parent2 and place it at the first position in parent1. Next the best matching neuron of the remaining neurons in parent1 with the second neuron of parent2 is placed at the second position. This matching process is continued for all the hidden neurons. Note that after the reordering the neurons in the last positions will not necessarily match very closely. The greedy reordering algorithm is only suboptimal and is a compromise between optimal matching and computational complexity. The suboptimal reordering should not be a problem however: in fact it introduces some diversity in the recombination process that might counteract premature convergence of the GA.

3.3 Hidden Node Redundancy Elimination

The functional redundancy at the individual hidden neuron level is very easy to eliminate. Whenever the number of positive incoming and outgoing weights of a neuron in the hidden layer is less than the number of negative incoming and outgoing weights, we simply flip their sign. This way we have reduced the 2^n functional equivalent neural networks to just one representative of the group.

4 Example: swinging up the inverted pendulum

4.1 problem description

To test the above ideas we apply the algorithm to find a feedforward neural network controller that has to swing up the inverted pendulum from its lower equilibrium point to its upper equilibrium point and stabilize it there. The design method used to accomplish this is proposed in [5]. The general idea is to use a feedforward neural network as a parametrized control law. The network has as input the four state variables (the position and velocity of the cart, and the position and velocity of the pendulum), and the output is the continuous control force acting on the cart and limited to a maximum allowed force. The control law represented by the neural net is overparametrized and constrained in the following sense: in the neighborhood of the upper equilibrium point, the control law has to coincide with a linear stabilizing controller around the target point. The additional freedom in the parameters is used to enforce the desired swinging up from the lower to the upper point.

Suppose we have a single input nonlinear system

$$\dot{x} = f(x, u)$$

with state vector x, input u and f a vector field. The control task is to bring the state x form the initial state x_o to the target state x_{eq}. We have to determine a nonlinear parametrized static state feedback law

$$u = g(x, w)$$

where w is the parameter vector [6]. In the case of a neural network controller these parameters are the connection weights. For a single layer feedforward network with one output neuron and tangent hyperbolic transfer functions the control law is given by:

$$u = F_{max}.tanh(\sum_{i=1}^{n_h} w_i.tanh(\sum_{j=1}^{n_{in}} v_{ij}.x_j))$$

with F_{max} the maximal allowed control force, n_h the number of hidden neurons, n_{in} the number of inputs, v_{ij} the weights from the input to the hidden layer, and w_i the weights from the hidden layer to the output neuron.

A state-space model of the inverted pendulum can be given by

$$\dot{x} = f(x) + b(x).u$$

with state x, input u and

$$f(x) = \begin{pmatrix} x_2 \\ \frac{\frac{4}{3}mlx_4^2 sinx_3 - \frac{mg}{2}sin(2x_3)}{\frac{4}{3}m_t - mcos^2 x_3} \\ x_4 \\ \frac{m_t g sinx_3 - \frac{ml}{2}x_4^2 sin(2x_3)}{l(\frac{4}{3}m_t - mcos^2 x_3)} \end{pmatrix}$$

$$b(x) = \begin{pmatrix} 0 \\ \frac{4}{3}.\frac{1}{\frac{4}{3}m_t - mcos^2 x_3} \\ 0 \\ \frac{-cosx_3}{l(\frac{4}{3}m_t - mcos^2 x_3)} \end{pmatrix}$$

The states x_1, x_2, x_3 and x_4 are respectively the position and the velocity of the cart, and the position and velocity of the pendulum. The symbol m is the mass of the pendulum, m_t the total mass of cart and pendulum, l is half the pendulum length and G the gravity constant.

The starting point is the lower equilibrium point $x_o = [00\pi 0]$ and the target state is the upper equilibrium point $x_{eq} = [0000]$. To swing up the pendulum we take as cost function

$$C = x_N^t x_N$$

with x_N the state vector that we have reached after a certain time period. In optimal control terminology this means that we are performing terminal control. Swinging the pendulum up however is not sufficient; we also want to stabilize it in its upper position. To achieve this the weights of the network are constrained by 4 equations so that the neural controller will coincide with a linear static state feedback controller (LQR) around the target point. The output of the linear controller is given by $u = -k_{lqr}^t.x$. A stabilizing controller around the upper equilibrium point can be achieved with a single neuron with weight vector

$$w = \begin{pmatrix} 0.1000 & 0.2303 & 3.1894 & 0.8178 \end{pmatrix}$$

$F_{max} = 10$ and

$$k_{lqr}^t = -F_{max}.w^t$$

[5].

To let the multilayer neural controller coincide with the linear controller we have to satisfy the four constraints

$$k_{lqr}^t = -F_{max}.w^t.v$$

4.2 experimental results

In the experiments we used a network with 4 hidden neurons. F_{max} and k_{lqr} are known so we can satisfy the constraints by computing the 4 output weights w form the 16 input weights v by simply inverting the input weight matrix. Although the output weights are also represented on the genotype the GA does not actually have to search them: before a newly created network is evaluated the output weights w are first computed from the input weights v. The values of the parameters in the experiments are $m = 0.1$, $m_t = 1.1$, $l = 0.5$ and $F_{max} = 10$. The genetic algorithm used is a steady state GA with a population size $n = 100$. Two parents are randomly selected and recombined following the approach outlined in the previous section - hidden neurons with their incoming and outgoing weights are exchanged as a whole. One of the offspring is evaluated by simulating it for 3 seconds and when it has a better function value than the worst of the current population it replaces this worst network. After every single recombination one individual is randomly picked out for a single hill-climbing step: one of the hidden neurons is selected and its weights are mutated by adding gaussian noise with zero mean and 0.1 variance. When the mutated network is better it replaces its parent, otherwise it is not included.

Figure 1 shows the results for 25 independent optimization runs of the neural controller with and without using the functional redundancy elimination. The curves represent the mean cost function value of the best network in the population after a certain amount of network evaluations. The lower curve is obtained when the hidden neurons are reordered before recombination and with the weight signs flipped. The upper curve is the result obtained when we simply recombine the genotypes without any reordering or sign flipping. The small vertical lines indicate the standard deviation for the lower curve.

Figure 2 shows a simulation of the swinging up of the pendulum by a typical neural controller. This network has a cost function value of 0.052 after 25000 function evaluations which is the median value of the

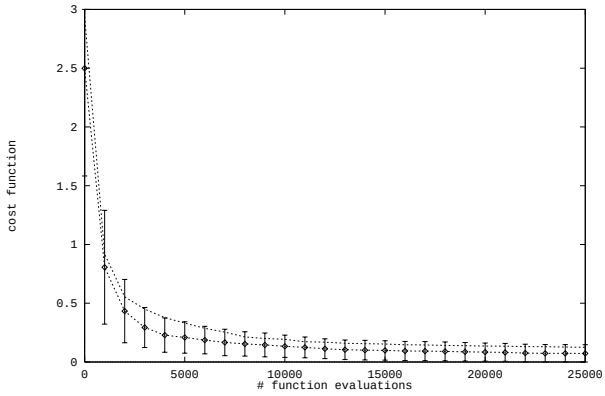


Figure 1: *Curves represent the mean - for 25 runs - cost function value of the best network in the population after the indicated number of network evaluations. The lower curve is obtained when the hidden neurons are reordered before recombination and with the weight signs flipped. The upper curve is the result obtained when we simply recombine the genotypes without any reordering or sign flipping. The small vertical lines indicate the standard deviation for the lower curve.*

25 independent runs when using the modified genotype representation. The weights of this controller are

$$v = \begin{pmatrix} 1.02824 & 1.14631 & 0.58466 & 0.54718 \\ 0.72937 & 2.09564 & 0.04480 & -0.25648 \\ -0.26217 & -0.37379 & 0.11679 & 0.06484 \\ 1.67519 & -1.19728 & 1.22857 & -1.67968 \end{pmatrix}$$

$$w = \begin{pmatrix} 1.78359 & 1.82575 & 13.95555 & 0.35406 \end{pmatrix}$$

The simulation clearly shows how the neural controller smoothly swings up the inverted pendulum and since the output weights w are constrained by the linear LQR controller the pendulum is stable at its upper equilibrium point. Of the 25 optimization trials with the functional redundancy elimination 23 of them were able to swing up the pendulum close enough to the target point so that a stabilizing controller was achieved. For the straightforward recombination only 16 trials were successful.

5 Discussion

The experiments show that the functional redundancy elimination gives better results: the mean value of the straightforward crossover is almost one standard deviation worse than the mean of the modified representation. This might at first seem a very

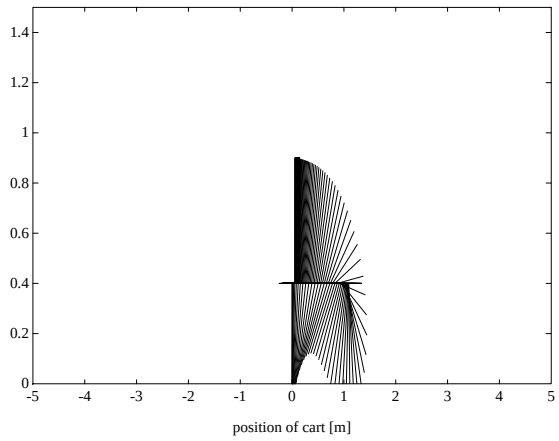


Figure 2: *simulation of the swinging up and stabilization of the inverted pendulum with the genetic optimized neural controller with median cost function value.*

little improvement but one should realize that the neural controller is actually a very small network. Since there are only 4 inputs, 4 hidden neurons and 1 output neuron there are just 16 possible weight sign combinations for a hidden node (1 with no inhibitory link, 5 with 1 inhibitory link and 10 with 2 inhibitory links). It can be expected that the genotype modifications will be more advantageous with increasing network size but this remains to be confirmed.

A second remark can be made on the nature of the GA search in general. When inspecting the solutions found it is clear that multiple weight combinations - aside from the symmetrical ones - are able to swing up the pendulum and stabilize it. Since we are using a simple GA with a small population size and a mixed in local hillclimber the search is very rapidly concentrated on one or very few promising regions - a phenomenon called "genetic hillclimbing" [7]. Considering the multimodal character of the cost function, techniques that promote niches to occur (e.g sharing [8]) are no doubt worth investigating.

A last remark concerns the efficiency of the search. The crossover operator as described here only exchanges complete hidden neurons with all their incoming and outgoing weights. The search for good hyperplanes itself is done by the gaussian hillclimber with constant variance. It is well known from the work of Evolution Strategies [9] that this is actually a very bad local search algorithm, so the performance will be much improved if a better local search technique is used, especially when second order gradient information is exploited.

6 Conclusion

The paper discusses the genetic weight optimization of a feedforward neural network. It is argued that the functional equivalent symmetries that exist in such networks cause a problem for the crossover operator. A way to eliminate these representation redundancies is to reorder the hidden neurons on the genotype before recombination according to a weight sign matching criterion and to flip the weight signs of a hidden neuron's connections whenever there are more inhibitory than excitatory incoming and outgoing links.

Experimental results were obtained for optimizing a neural controller that has to swing up the inverted pendulum and stabilize it at the upper equilibrium point.

References

1. Hornik K., *Multilayer Feedforward Networks are Universal Approximators*, Neural Networks, Vol.2, pp.359-366, 1989.
2. Funahashi K.I., *On the Approximate Realization of Continuous Mappings by Neural Networks*, Neural Networks, Vol.2, pp.183-192, 1989.
3. White H., *Connectionist Nonparametric Regression: multilayer feedforward network can learn arbitrary mappings*, Neural Networks, Vol.3, pp.535-549, 1990.
4. Sussmann H.J., *Uniqueness of the Weights for Minimal Feedforward Nets with a Given Input-Output Map*, Neural Networks, Vol.5, pp.589-593, 1992.
5. Suykens J. & De Moor B., *Stabilizing Neural Controllers with Application to the Control of an Inverted Pendulum*, ESAT-SISTA report nr.16, K.U.Leuven, 1992.
6. Bryson A.E. & Ho Y.C., *Applied Optimal Control*, Waltham, Ma. Blaisdel, 1969.
7. Whitley D., Starkweather T. & Bogart C., *Genetic Algorithms and Neural Networks: optimizing connections and connectivity*, Parallel Computing 14, pp.347-361, 1990.
8. Deb K. & Goldberg D.E., *An Investigation of Niche and Species Formation in Genetic Function Optimization*, in Schaffer J.D.(ed.) Proceedings of the Third International Conference on Genetic Algorithms. San Mateo, CA. Morgan Kaufmann Publishers, 1989.
9. Schwefel H.P., *Numerical Optimization of Computer Models*, Wiley, Chichester, 1981.