Topology design of feedforward neural networks by genetic algorithms

Slawomir W. Stepniewski[1] and Andy J. Keane[2]

[1] Department of Electrical Engineering, Warsaw University of Technology, ul. Koszykowa 75, 00-662 Warszawa, Poland, E-mail: step@bsdi.iem.pw.edu.pl.

[2] Department of Mechanical Engineering, University of Southampton, Highfield, Southampton, SO17 1BJ, U.K, E-mail: andy.keane@soton.ac.uk

Abstract. For many applications feedforward neural networks have proved to be a valuable tool. Although the basic principles of employing such networks are quite straightforward, the problem of tuning their architectures to achieve near optimal performance still remains a very challenging task. Genetic algorithms may be used to solve this problem, since they have a number of distinct features that are useful in this context. First, the approach is quite universal and can be applied to many different types of neural networks or training criteria. It also allows network topologies to be optimized at various level of detail and can be used with many types of energy function, even those that are discontinuous or non-differentiable. Finally, a genetic algorithm need not be limited to simply adjusting patterns of connections, but, for example, can be utilized to select node transfer functions, weight values or to find architectures that perform best under certain simulated working conditions. In this paper we have investigated an application of genetic algorithms to feedforward neural network architecture design. These neural networks are used to model a nonlinear, discrete SISO system when only noisy training data are available. Additionally, some incidental but nonetheless important aspects of neural network optimization, such as complexity penalties or automatic topology simplification are discussed.

1. Introduction

When designing feedforward neural networks for the purpose of nonlinear approximation, typically architectures with one or two hidden layers are used. Because there is no precise guidance on how to choose the number of nodes in each layer or the size of the whole network, designing such a structure is usually a trial and error procedure. For a given problem many different network architectures may be able to reconstruct a desired input-output mapping with similar error levels. Moreover, using cross-validation techniques, it is possible to train even oversized architectures, since this kind of approach is designed to avoid overfitting [6]. For this reason, the application of neural networks may seem to be quite simple and straightforward. If no serious mistakes are made and the problem is suitable for modelling by a feedforward neural network (the function being approximated is for example smooth, continuous and has bounded values), many configurations should work fairly well despite wide variations in their sizes, types and interconnection patterns. On the other hand, however, a designer usually wants to be assured that the proposed network is optimal or close to optimal in some sense. Also, in some cases, maximal performance may be very desirable. This type of problem arises for example in control schemes that utilize basic or inverse plant models. The controlling capabilities of such schemes improve as the synthesized model better matches the object. Automated design methods start to become attractive in such situations. Here, we present several new results concerning the application of genetic algorithms to the automated design of network topologies.

2. Initial Network Architecture

Before applying a genetic algorithm to neural network topology optimization, information about the network architecture must be converted into an appropriate format that can be processed by the method. Binary strings are used most often with genetic algorithms, although it is also possible to utilize custom designed encodings together with relevant recombination and mutation operators [9]. Here, we have adopted binary chromosomes that are built from the elements corresponding to network synapses only. A value of one indicates that a link exists, zero means that the connection is removed. All hidden and output nodes are always biased. Because it is not desirable to delete biases, bits indicating their presence are not included in the encoding.

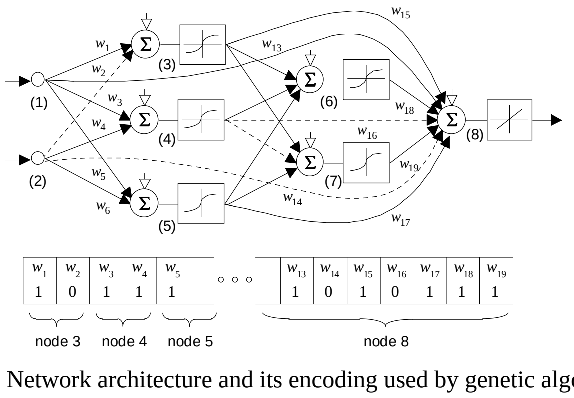
Figure 1 illustrates the type of network considered in this paper. All hidden units have sigmoid ($f(x)$=tanh(x)) transfer functions. The output node has a linear ($f(x)=x$) transfer function. The initial topology to be pruned is a two layer architecture with additional direct connections from each hidden and input node to the network output. This kind of topology was chosen for the following main reasons:

1. Additional links that skip adjacent layers allow the genetic algorithm to remove a whole layer of nodes while still keeping the network functional.
2. Some additional shortcut connections, e.g., those between the network input and output [12], may ease training and therefore the whole genetic topology optimization may become faster.
3. The architecture adopted has a better capability to realize mappings with linear trends, due to the direct links between the input and output layer (for training we have used raw data without any preprocessing to remove DC components or linear trends).
4. Because all the networks pruned here have only one output, the length of the chromosome string does not increase significantly; it rises by the total number of hidden and input units in comparison to a layered architecture with only adjacent units connected.

Other choices for the initial neural network topology are discussed in [11].

The choice of bit locations in the chromosome is not addressed in detail here. It is worth noting, however, that the bits associated with one node are initially bound together but the genetic algorithm applied uses inversion to alter gene ordering and attempts to discover arrangements that help to preserve valuable building blocks.

It should be noted that the binary encoding used here may be difficult to apply to large scale networks (e.g. >1,000 links) but for modelling applications encountered in control



w_1	w_2	w_3	w_4	w_5	∘ ∘ ∘	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	w_{19}
1	0	1	1	1		1	0	1	0	1	1	1

Figure 1: Network architecture and its encoding used by genetic algorithm.

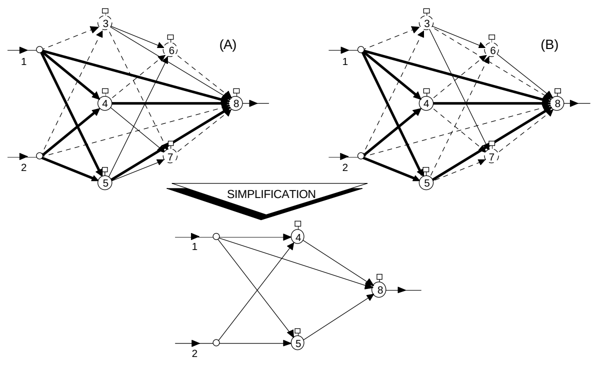


Figure 2: An example of the topology simplification procedure. The bold lines of networks (A) and (B) represent active connections, hairlines correspond to the passive links and dotted lines show deleted synapses.

engineering it is rather unusual to construct neural networks with more than 200 or 300 free parameters (weights and biases). Besides, this kind of encoding allows a search space to be sampled quite precisely and can be used with almost any genetic software package. Any kind of genetic operation always produces a valid neural network (we assume that the architecture is valid if it does not violate the principle of the feedforward transmissions of signals, i.e., there are no explicit closed loops inside the network, althought it is permitted for some input signals not to reach the network outputs). Of course, with this kind of encoding, it is easy to create architectures that obviously will not work correctly, e.g., networks that do not process any input signals. We have never found this possibility to be a real problem; when such configurations occur, this should rather be treated as an indication that some parameters of the topology optimization are adjusted incorrectly (e.g., an excessive penalty on network complexity may give rise to this kind of behaviour) or the task to be implemented by the network is trivial so very few nodes are sufficient to perform the desired function approximation. A genetic algorithm is then likely to sample extremely small networks including those with no active synapses.

3. Topology Simplification

It is clear that topologies generated by a genetic algorithm may contain many superfluous components such as single nodes or even whole clusters of nodes isolated from the network input. Such units (we call them passive nodes) do not process the signals applied to the network sensors and produce only constant responses that are caused by the relevant biases. These passive nodes, if finally connected to active nodes (those transmitting network input signals), function simply as extra biases (e.g., the connection between node 3 and 8 of network A, figure 2). A simplification procedure (a similar idea was used for example in [7]) can therefore be used to remove passive nodes and links and modify true biases so that the input/output mapping of the network remains unchanged. In addition, the simplification procedure deletes all totally isolated links and nodes that have no influence on the network output (e.g., the connection between nodes 5 and 7 of network A, figure 2). A genetic algorithm has no chance to perform this type of simplification efficently by itself as the existance of spare connections has little or no influence on the value of the fitness function. A detail diagram of the simplification algorithm used here is presented in figure 3.

In our implementation of the topology search, the simplification procedure is always carried out before the network is trained. This leads to a reduced number of parameters (weights and biases) to be adjusted by the training algorithm and so learning is typically performed faster. On the other hand, however, training of the simplified networks may proceed in different ways to those of their raw counterparts. This can occur because of the different number of parameters involved, for example. The starting point of the training can also be changed because the simplification procedure modifies certain biases in order to preserve the current network activity. It is interesting to note that a particular, simplified network may be obtained from several, different parent architectures (an example of such a situation is presented in figure 2). As the starting points for training may be different for such networks so also may be their final performance and thus their estimated usefulness. Sumarizing, we can say that two or more different chromosomes corresponding to the same simplified architecture may be assigned various fitness values due to different network performance achieved.

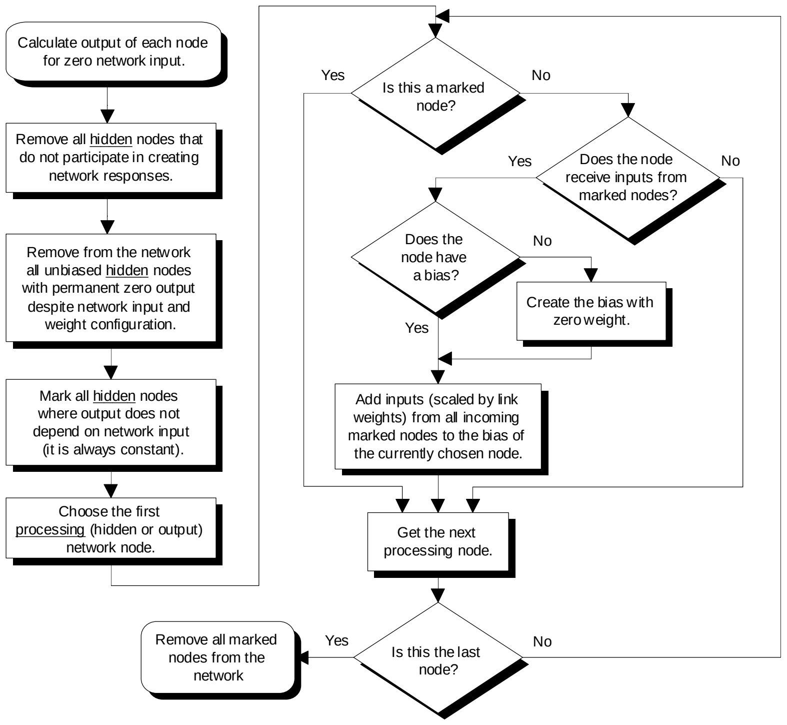
The topologies obtained after simplification are used to asses the real (effective) complexity of the encoded network. Obviously, this task cannot be accomplished by simply calculating the number of '1's in the chromosome because some bits may refer to passive links. Figure 2 is based on such an example of two different size, raw architectures that are functionally equivalent; for network A, five existing links are redundant while network B has only three such connections.

4. Objective Function

Using a genetic algorithm for network topology optimization requires an objective function to be constructed which allows comparison of the usefulness of the network architectures examined (in fact, this approach could be treated as hierarchical optimization with one stage involving network training and the second one being connection pruning). The objective function may combine several factors that reflect the overall quality of a particular network architecture, e.g., number of nodes, number of synaptic connections, learning accuracy, generalization ability and so on. Some components of the objective function, such as generalization ability cannot be evaluated precisely. They depend heavily on many hidden factors that are difficult to quantify by a simple formula, e.g., the choice of training algorithm, its internal parameters and starting point, the effectiveness of training data, termination condition, etc. Other parameters associated with network complexity (number of weights, biases, nodes or inputs) can be calculated directly but converting this information into a useful coefficient that robustly describes network topology is also not trivial (to help in this process we have used the results provided by the topology simplification process discussed ealier).

The fundamental parameter that reflects the usefulness of a feedforward network with analog output remains, however, its generalization ability. Although this factor cannot be evaluated easily it should form a major part of the objective function. Despite the relatively high cost of estimating generalization (it of course involves training each network) some of its properties still make assessing and comparing network quality difficult.

- First, generalization is evaluated with finite accuracy and therefore some differences between networks remain unrevealed. The maximum level of generalization is constrained, for example, by the target learning error (assuming this error level is reachable for a given training algorithm or the maximum number of training



Calculate output of each node for zero network input.
Remove all hidden nodes that do not participate in creating network responses.
Remove from the network all unbiased hidden nodes with permanent zero output despite network input and weight configuration.
Mark all hidden nodes where output does not depend on network input (it is always constant).
Choose the first processing (hidden or output) network node.
Is this a marked node?
Yes
No
Does the node receive inputs from marked nodes?
Yes
No
Does the node have a bias?
No
Yes
Create the bias with zero weight.
Add inputs (scaled by link weights) from all incoming marked nodes to the bias of the currently chosen node.
Get the next processing node.
Is this the last node?
Yes
No
Remove all marked nodes from the network

Figure 3: A general flowchart of the simplification routine that can be applied to networks built using both biased and unbiased nodes.

iterations). Since training is always stopped when the network error achieves some required accuracy, typically the generalization error per one data sample should not be lower than the preset, target training error.

- Second, the generalization ability of a given network is estimated after training. The intermediate step of training allows different networks to be adjusted to perform a requested function approximation with similar degrees of performance. The longer and more efficient the training algorithm is, the greater the portion of the networks examined that achieve a given accuracy level. This phenomenon means that when more precise training is used, a genetic algorithm may face difficulties distinguishing between various design options and extracting features that help to build superior networks (it is easier to recognize badly functioning outliers rather than to identify exceptionally valuable individuals). On the other hand, using crude network evaluation may also be inappropriate since no useful information may be provided to the topology optimizer except parameters that are obvious, i.e., network size. Although the genetic algorithm is known as a robust optimization method that can function in noisy environments, allowing inaccurate training may mean that the final 'winners' of the network topology optimization will not achieve the expected performance when subsequently trained more precisely.
- Finally, generalization assessment is affected by many types of disturbances that cannot be totally compensated for. Any fluctuation of the training starting point, or the number of data points may influence the training algorithm. Also, the internal

settings of the weight adjusting algorithm or even such 'remote' parameters as the initial learning coefficient may alter the whole training process considerably. Therefore, there is no simple and exact relationship between neural network topology encoding (genotype) and its performance and usefulness for the particular task (phenotype).

In our experiments we have used an objective function based on the form:

$$F(\cdot) = \left(\frac{E_V}{N_V} + \frac{E_L}{N_L} \right)^R \left(1 + \frac{L_A}{L_{\max}} \right)^S \tag{1}$$

where E_V, E_L are the sums squares of validation (post-training testing) and training errors, respectively. N_V and N_L refer to the number of data samples used during the validation and training phases. L_A is the number of connections after network simplification, $L_{\max}$ is the maximum number of allowed links (the length of the chromosome used to encode the initial topology) and *R*, *S* are fixed parameters. Our experiments indicate that these parameters may sometimes significantly affect the performance of the genetic algorithm. For the experiments presented in this paper we have used $S = 0.5$ and $R = 1$ which seemed to work reasonably well.

The fitness function defined by formulae (1) directs the topology search toward networks that (i) learn the desired patterns well, (ii) generalize correctly and (iii) have the smallest possible size.

Here, incorporating the learning error in the objective function is not so important as the validation error because the learning error is on average much smaller. On the other hand, however, the value of E_L also carries some information about network performance and we treat it as a 'second order' correction factor. The difference between these two components depends on many aspects of the training process, e.g., the stopping criteria. Employing the cross-validation technique usually prohibits the learning error from having very small values while allowing the validation error to be rather higher. In our test, when the cross-validation method was used, E_V was usually about 4 to 5 times higher than the learning error E_L. Nonetheless, we decided to keep the learning error in the objective function (1) to assure that the networks that fail to train correctly or those which produce rare results when $E_V < E_L$ have a high objective function and are then likely to be rejected by the genetic algorithm[1].

The term $(1 + L_A / L_{\max})$ in the objective function (1) can be interpreted as a complexity penalty factor. In some cases, especially when the initial, unpruned network is small, this penalty factor may be difficult to adjust precisely using the parameter *S*. The penalty function may still overly reward small networks and remove too many links. In such cases, a modified complexity penalty factor Θ can be applied:

$$F(\cdot) = \left(\frac{E_V}{N_V} + \frac{E_L}{N_L} \right)^R \left[1 + (\gamma - 1) \left(\frac{L_A}{L_{\max}} \right)^S \right] = \left(\frac{E_V}{N_V} + \frac{E_L}{N_L} \right)^R \Theta\left(L_A / L_{\max} \right) \tag{2}$$

---

[1] Our version of the genetic algorithm utilizes an extinctive selection scheme so that the worst chromosomes are always removed from consideration in the process of creating a new population.

The parameter γ ($\gamma \geq 1$) affects the maximum value of the complexity penalty while the exponent *S* controls the shape of the function Θ: for *S* approaching unity the penalty is close to linear but as *S* increases a stronger penalty is applied to the biggest networks leaving smaller and medium size architectures relatively unpenalized.

5. Genetic Algorithm

The genetic algorithm tested here uses all four standard operators: crossover (crossover probability, $P_C = 0.8$), mutation ($P_M = 0.01$), inversion ($P_I = 0.2$) and selection. The selection[2] scheme is generational, linear ranking and extinctive, allowing the best 80% of the current population to breed and discarding the remaining 20% of chromosomes (this allows those chromosomes that correspond to the networks that were not trained successfully to be rejected). The selection operator is also 1-elitist so that the best performing individual of each generation is assured of being included in the next population unchanged. To prevent the genetic algorithm from being dominated by a few moderately good designs that may block further exploration of the search space a fitness sharing scheme is employed as described in [5, 8]. This method performs a cluster analysis of the current population and modifies the raw objective function so that the chances of creating individuals in overcrowded regions are reduced. A somewhat opposite operation, inbreeding, is launched if, despite the niche penalty, a particular cluster remains numerous.

For all the experiments presented in this paper the population size was chosen to be 50 and the genetic algorithm was run for 10 generations. Because the genetic algorithm reuses some top performing designs when a new population is created, the actual number of different network evaluation is slightly less than 500; typically it was 5 to 8% lower.

6. Experimental Results

We have applied the the pruning technique as described above to the problem of designing a neural network that is employed to model a discrete time, noisy, non-linear system. The neural network is treated as a one step predictor. The system to be identified has one input and one output (SISO class). Its behavior at discrete points may be described by a non-linear equation of the form:

$$y(k) = G\big(y(k-1),\ldots,y(k-n_a),u(k-1),\ldots,u(k-n_b),e(k),\ldots,e(k-n_c)\big) \qquad (3)$$

where $y(k-1),\ldots,y(k-n_a)$ are past system outputs, $u(k-1),\ldots,u(k-n_b)$ are relevant system inputs and $e(k),\ldots,e(k-n_c)$ is a disturbance sequence. Because feedforward neural networks propagate signals with no delays, to emulate the dynamics of a non-linear system the MLP is fed with several, time shifted signals that come both from the system input and/or output. Here, we consider an idealized situation when the number of network inputs is chosen to match the mathematical description of the modelled plant. The plant behaviour is described by the following discrete equation:

$$y(k) = \frac{y(k-1) + 0.5u(k-1) + 3.6y(k-1)u(k-1) + y(k-1)e(k)}{1 + y^2(k-1) + u(k-1)e(k)} - 1.5 \qquad (4)$$

[2] The terminology used to describe selection follows the definitions presented by Bäck and Hoffmeister in [2].

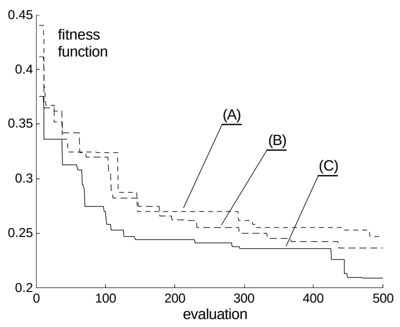


Figure 4: Average performance of a genetic search for increasing sizes of validation data pools: A - 100 samples, B - 400 samples, C - 800 samples.

where $e(k)$ is non-additive, normally distributed white noise of variance 0.05. This is a modified version of the problem that was investigated in [3]. The plant was excited with a uniform, zero mean signal $u(k)$ ranging between -2.0 and 2.0. All networks were trained using the RPROP [10] algorithm. The number of training data was 100 (30% of which were used for cross-validation); three sets of post-training, verification data were used with 100, 400 and 800 samples. The initial network topology to be trimmed was 2-6-6-1 with 62 connections and 13 biases.

When a plant is affected by significant noise that cannot be filtered out, some kind of noise estimator may be established to make prediction more accurate and this is identified along with the main object model (see [1] for a tutorial treatment of this subject). Also, the influence of noise can be reduced by modification of the identification algorithm and utilization of model outputs $\hat{y}(k)$ as regressors instead of the real plant values, but this kind of identification approach (instrumental variable method or output-error model) requires a more complicated iteration process for finding model parameters and larger amounts of training data to obtain valid results However, our task here is not to create a good predictor *per se* but to investigate the performance of the genetic algorithm for network topology design when neural networks are trained using data corrupted by noise. Therefore, the approach used in constructing the neural models does not utilize noise estimation or modified network training.

Optimizing neural network topology when noisy training and validation data are used creates a new situation for the genetic algorithm. Seemingly, there is no longer a lack of diversity in the problem because of the stronger fluctuations in network training and performance. However, this will not help a genetic search, since values of the fitness function are likely to be erroneously evaluated. Indeed, when neural networks were trained and verified on small numbers of data points, the genetic algorithm did not converge: after creating the initial population, usually, no improvements were encountered or successful steps were isolated and rare. We may suspect that the final results produced were flukes rather than the effect of the choosen search strategy. This situation starts to improve as we increase the quantity of training and validation data. Obviously, the training pool cannot be expanded without limit since it slows down the whole topology optimization. It is, however, possible to test inexpensively a network, even on a very large data sets, after completing training. We have found that utilizing larger post-taining validation tests steadily improves the performance of a genetic algorithm in finding good designs (figure 4).

Method/Ratio	Objective function	Generalization	Number of links
GA - search	2.067095e-01	2.013831e-02	20
enumeration	3.893958e-01	5.939192e-02	35
GA/enumeration ratio	0.53	0.34	0.57
GA - search	2.009306e-01	1.430509e-02	30
enumeration	3.319947e-01	5.892727e-02	19
GA/enumeration ratio	0.61	0.24	1.58
GA - search	2.212368e-01	2.730137e-02	14
enumeration	2.710268e-01	1.717330e-02	53
GA/enumeration ratio	0.81	1.59	0.26
GA - search	2.048909e-01	1.781993e-02	24
enumeration	4.095443e-01	6.307428e-02	35
GA/enumeration ratio	0.50	0.28	0.69
GA - search	2.117007e-01	1.978875e-02	22
enumeration	2.662749e-01	2.509494e-02	35
GA/enumeration ratio	0.80	0.79	0.63

Table 1: Comparison of the best networks for problem (4) found by a genetic algorithm and by sequential evaluation.

So far we have focused on the performance of the genetic algorithm alone. However, an important question is whether it is worth applying this method as an alternative to a simple trial and error approach that can also lead to successful solutions. For this reason, we have finally compared the performance of genetically optimized neural networks with manually crafted designs. The latter approach was emulated by a sequential evaluation of fully connected architectures obtained after removing increasing numbers of hidden nodes from the initial topology used to start the genetic algorithm. For the initial architecture 2-6-6-1 this led to training 36 different networks (topologies: 2-6-6-1, 2-6-5-1,...,2-6-1-1, 2-5-6-1, 2-5-5-1, and so on) in 5 series (each one started from different random seeds) and a comparison, according to the fitness function, of the best result found by both methods. Tables 1 summarizes these results.

For the problems studied here the genetic algorithm was *always* able to find a better neural network topology with respect to the chosen fitness function. In most cases both smaller size and better generalization ability could be achieved, although in some instances only one of these parameters has superior fitness. The results of creating neural models of the noisy plant are promising as they suggest that genetic algorithms may be a useful tool for solving these difficult problems. The results obtained also reveal that architectures with problem specific patterns of connections may indeed perform better than more restrictive, fully connected topologies arranged into geometrically regular layers. Besides, their size as measured by the number of connections may be expected to be smaller as well.

7. Conclusions

In this paper we have presented a network topology optimization method that is based on using a genetic algorithm. Various aspects of this technique, such as topology simplification, complexity assessment and the impact of network training on a genetic search have been discussed. Although, the simplification procedure was integrated quite closely with a topology optimizer, the same method of removing inactive or redundant connections may also be useful when other pruning techniques like OBS or OBD [6] are used and the initial feedforward architecture has connections that skip adjacent layers.

The method presented is CPU expensive due to the exceptionally laborious process of neural network training. This time consuming procedure requires a designer to choose carefully the number of training data sets and the initial, unpruned topology to avoid unnecessary computations. We have stressed that precise training may not guarantee ultimate success in finding superior architectures. A proper construction of the fitness function is an equally important (if not crucial) factor. Also, post-training validation is an essential part of network evaluation, as it detects not only the precision of weight adjustment but also the adequacy of the model structure. It is also worth noting that, although in the present study we have treated neural networks as one step predictors, the results presented in [4] suggest that utilizing them as multiple-step ahead predictors (autoregressive models only) may reveal topological deficiencies more clearly.

The genetic algorithm investigated here has proven to be a robust optimization technique, capable of operating in noisy environments while still producing superior results than could be obtained by manual, trial and error approaches. The genetic algorithm cannot be applied blindly, however. Although the method is well behaved and the wide range of investigations conducted elsewhere confirm that it does not need very precise tuning of its parameters, a good knowledge of the underlying problem modelling is definitely required.

References

[1] Ästrom K. J., Wittenmark B., Computer Controlled Systems - Theory and Design, Prentice-Hall, 1984.

[2] Bäck T., Hoffmeister F., Extended Selection Mechanisms in Genetic Algorithms, Proc. 4th Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 1991.

[3] Billings S.A., Zhu Q.M., Nonlinear model validation using correlation tests, Int. J. Control, 1994, vol. 60, no. 6, pp. 1107-1120.

[4] Billings S.A., Jamaluddin H.B., Chen S., Properties of neural networks with applications to modeling non-linear dynamical systems, Int. J. Control, 1992, vol. 55, no. 1, pp. 193-224.

[5] Goldberg D. E., Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, 1989.

[6] Haykin S., Neural Networks - A Comprehensive Foundation, Macmillan College Publishing, 1994.

[7] Kitano, H., Neurogenetic learning: an integrated method of designing and training neural networks using genetic algorithms, Physica D 75, 1994, pp. 225-238.

[8] Keane A.J., The Options Design Exploration System, Reference Manual and User Guide, 1994, available at http://www.eng.ox.ac.uk/people/Andy.Keane/.

[9] Kuscu I., Thornton C., Design of Artificial Neural Networks Using Genetic Algorithms: review and prospect, Cognitive and Computing Sciences, University of Sussex, 1994.

[10] Schiffmann W., Joost M., Werner R., Optimization of the Backpropagation Algorithm for Training Multilayer Perceptrons, Technical Report, University of Koblenz, 1992.

[11] Stepniewski S.W., Keane A.J., Pruning backpropagation neural networks using modern stochastic optimization techniques, (accepted for publishing by Neural Computing & Applications).

[12] Whitley D., Starkweather T., Bogart C., Genetic Algorithms and Neural Networks: optimizing connections and connectivity, Parallel Computing 14, 1990, pp. 347-361.