IMPROVING GENERALIZATION BY USING GENETIC ALGORITHMS TO DETERMINE THE NEURAL NETWORK SIZE

George Bebis and Michael Georgiopoulos

Department of Electrical & Computer Engineering
University of Central Florida
P.O. Box 25000, Orlando, FL 32816, U.S.A.
e-mail: {geb,mng}.engr.ucf.edu

Abstract

Recent theoretical results support that decreasing the number of free parameters in a neural network (i.e., weights) can improve generalization. The importance of these results has triggered the development of many approaches which try to determine an "appropriate" network size for a given problem. Although it has been demonstrated that most of the approaches manage to find small size networks which solve the problem at hand, it is quite remarkable that the generalization capabilities of these networks have not been explored thoroughly. In this paper, we propose the coupling of genetic algorithms and weight pruning with the objective of both reducing network size and improving generalization. The innovation of our approach relies on the use of a fitness function which uses an adaptive parameter to encourage the reproduction of networks having good generalization performance and a relatively small size.

1. Introduction

Network size in the case of layered, feed-forward, neural networks depends on the number of layers and the number of nodes per layer but usually is expressed in terms of the number of connections in the network. In general, network size affects network complexity, and learning time, but most importantly, it affects the generalization capabilities of the network; that is, its ability to produce accurate results on data outside its training set [1],[2]. A network having a structure simpler than necessary cannot give good results even for patterns included in its training set. On the other hand, a more complicated than necessary structure, "overfits" the training data, that is, it performs nicely on patterns included in the training set but performs very poorly on unknown patterns.

Major emphasis has been given in the last few years in the development of techniques which try to improve generalization by modifying not only the connection weights but also the network structure as training proceeds. These techniques can be divided into three main categories: *pruning*, [3]-[7], *constructive* [8],[9], and *weight sharing* [10],[11]. Although one of the most important reason for modifying the network architecture is to improve generalization, most of the existing approaches have not emphasized this issue. In most cases, the feasibility of an approach is illustrated showing results on network size reduction and convergence speed, while less or no emphasis has been given to the generalization issue [3]-[5],[8][9]. Actually, in some studies where generalization has been addressed, improvements were observed using only artificial data [6],[12],[13], while no significant generalization improvement or even worst generalization has been reported in other studies where real data were used [14]-[16]. In this paper, our focus is on the area of weight pruning techniques.

Weight pruning techniques are very sensitive to the selection of certain parameters which determine when pruning should start and when it should stop. If pruning starts too early, the network might not be able to learn the desired mapping. On the other hand, starting pruning late might waste training time and the network might start memorizing the training set. Also, if pruning stops early, we might not be able to sufficiently prune the network and this might lead to poor generalization again. Finally, if pruning does not

stop at the right time, then the network might be able to fit the training data satisfactorily, but it might not be able to generalize well. Usually, determining appropriate pruning parameter values to control the beginning and end of the pruning process is done by trial and error.

In this paper, we propose the coupling of genetic algorithms [17] and weight pruning. Research on combining genetic algorithms and neural networks has attracted a lot of attention lately [18]. The weight elimination technique [7], a representative weight pruning technique and the most general of weight decay approaches has been chosen to be coupled with the genetic algorithm. The purpose is not only to prune oversized networks but also to improve generalization and to make pruning less sensitive to the selection of the pruning parameter values. The innovation of our approach relies on the use of a fitness function which takes into consideration both the network size and the generalization performance. During the generation of new genetic populations, an adaptive parameter weighs appropriately the importance of network size versus generalization, encouraging the reproduction of networks having good generalization capabilities and small size.

The organization of the paper is as follows: Section 2 reviews the weight elimination technique. Section 3 discusses the genetic algorithm approach. The databases used and preliminary experimental results obtained are described in Section 4. Finally, our conclusions are given in Section 5.

2. Determining network size using weight elimination

Weight elimination minimizes the following modified error function:

$$E_{WE} = E_0 + \lambda_{WE} E_1 = \sum_k (target_k - output_k)^2 + \lambda_{WE} \sum_{i,j} \frac{w_{ij}^2/w_0^2}{1 + w_{ij}^2/w_0^2}$$

The first term is the original error function which is simply the sum of the squared errors between the actual output values and the desired (target) output values. The second term depends on the size of the network and λ_{WE} is a weighting factor which determines the importance of the second term with respect to the first. The choice of w_0 determines the number and magnitude of weights. If w_0 is chosen to be small, the algorithm converges to a solution having a few large weights. On the other hand, if w_0 is chosen to be large, a solution is obtained having many small weights.

An important issue to be addressed is when to start pruning. Specifically, we start removing connections only when the generalization performance of the network is satisfactory. A common way of testing generalization, is using cross-validation and this the approach used here [1]. Defining the generalization performance G_{val} as the ratio of correctly classified patterns from the validation set over the total number of patterns in the validation set, we start removing connections when $G_{val} > T$, where T is a threshold to be defined. A weight w_{ij} is removed only if $|w_{ij}| < |w_0|$.

Another important issue to be addressed is when to stop training and as a consequence, pruning. A robust way is using the average generalization error A_n defined as follows:

$$A_n = \gamma A_{n-1} + (1 - \gamma)(1 - G_{val})$$

where n denotes the epoch at which A_n is computed, A_0 is set to 1, and γ is a constant, to be chosen relatively close to 1. If A_n keeps decreasing, we continue training and pruning, otherwise, we stop. The update of G_{val} and A_n takes place every epoch.

Weigend et. al. [7] have proposed an adaptive procedure for determining the parameter λ_{WE}. Initially, λ_{WE} is set to zero. Then, it increases, decreases, or stays the same according to certain criteria. We have found this procedure to be quite heuristic and parameter dependent. Here, we have chosen an alternative way for determining λ_{GE} which has been motivated by [6]. Specifically, λ_{GE} is determined as follows:

$$\lambda_{GE} = \lambda_{0\_GE} e^{-\beta_{WE}(1 - G_{val})}$$

where λ_{0_GE} and β_{WE} are constants to be defined. The updation of λ_{WE} takes place every epoch.

3. Evolving neural network connectivity

Genetic algorithms operate iteratively on a population of structures, each one of which represents a candidate solution to the problem at hand. An important issue is how the architecture should be represented (encoded) into a structure (string of symbols) that can be handled by the genetic algorithm. Once an encoding scheme has been chosen, a number of networks are encoded to form the initial population. On each iteration, a new population is produced by first applying on the old population a number of genetic operations. Then, each member of the population is evaluated through a fitness function, assuming first that each member is decoded into a legitimate network architecture. Members that performed badly are discarded, while members that performed well survive in future populations. Finally, the genetic algorithm converges to a population whose best member represents the solution the algorithm has found.

The proposed approach tries to prune oversized networks with the objective that the obtained pruned networks will always have a smaller size and a better generalization performance than their unpruned counterparts. Initially we chose a two-layer network (i.e., one hidden and one output layer), with enough nodes in the hidden layer to ensure convergence. The reason we have restricted ourselves to two-layer networks is because of a very important theoretical result stating that *a single hidden layer feedforward network with arbitrary sigmoid hidden layer activation functions can approximate arbitrarily well an arbitrary mapping from one finite dimensional space to another* [19]. After an oversized network has been chosen, we encode it into a structure that can be handled by the genetic algorithm and we create P copies of it. P should be the number of members in a population (population size). Each of these copies is assigned a different set of parameter values. The parameters were chosen to be the initial weights, β_{WE}, and w_0. This choice was based on some preliminary experimental results which indicated that these parameters affect generalization and network size the most. New populations are generated by applying the genetic operators of reproduction, crossover and mutation. Then, the fitness of each member is measured by first decoding it into a network and then training it for a number of epochs using weight elimination in order to record the network's performance in terms of generalization and size.

Each network has its own parameter λ_{WE} which weighs generalization versus network size, its own generalization performance G_{val}, and its own average generalization error A_n. This means that after some generations members in the same population should have totally different characteristics and this is the motivation for using different initial weights, β and w_0 for each member of the initial population; by creating a large number of different size networks having various generalization performances, allows the algorithm to discover better solutions. It should be emphasized that pruning does not occur only because of weight elimination during evaluation but also because of the use of the crossover operator. Thus, the solutions obtained by combining the genetic algorithm with weight elimination can not be replicated using weight elimination by itself.

3.1. Network representation scheme

A popular representation scheme has been proposed by Miller et. al. [20], where the network architecture is represented as a connection matrix, which is mapped directly into a bit-string. Although this scheme seems to satisfy our requirements, it has the disadvantage that it creates very long strings. Here, we have adopted another approach proposed by Montana and Davis [21]. According to their approach, the weights and biases of a network are encoded in a straightforward way as a string of real numbers. Decoding is again straightforward. In addition, since certain connections are eliminated during evolution, each connection is not only associated with a weight value but also with a flag, indicating if a connection exists (flag=1) or not (flag=0). Initially, all the flags are set to one. As new populations are produced as weight elimination takes place, some of the connections have their flags set to zero to indicate that they have been pruned.

3.2. Genetic operators

The genetic operators which have been used in this work, are the most commonly used operators: reproduction, crossover, and mutation. The purpose of the reproduction operator is to create a new population based on the evaluation (fitness) of the members of the old population. Each member of the old population produces a number of exact copies such the the most fit members produce the most copies. Our implementation uses the roulette wheel selection scheme described in [17]. Fitness scaling has been also implemented to avoid premature convergence. In addition to fitness scaling, two more heuristics have been incorporated in our implementation: the generation gap and the elitism strategy [17].

Crossover is applied after reproduction. Traditionally, crossover works as follows: pairs of members are selected at random and portions of them are exchanged to form new members. Here, we are using a modified crossover operator which is called the crossover-nodes operator [21]. The idea is to swap groups of weights feeding into the same node. The reason is quite plausible; each node in the network contributes to the solution the network tries to find. Thus, weights feeding into a node serve a role in finding a solution for the problem at hand. Swapping weights arbitrarily may not make a lot of sense while swapping groups of weights feeding into nodes is more sensible.

The last genetic operator used is the mutation operator. This operator picks a member of the population randomly and changes it slightly. In its simplest form, mutation changes the value of a weight by adding a small random value. Following our discussion regarding the crossover-nodes operator, the mutation operator we used in this study, does not change not single weights but groups of weights feeding into a same node. This modified operator which we call the mutate-nodes operator, has also been used in other studies [21], illustrating good performance.

3.3. Fitness evaluation

The most commonly used approach to evaluate a network's performance is to train the network represented by a member in the population and record the network's mean square error. However, this is quite inefficient for our purpose, since it does not account for the network's generalization performance and size. To perform an evaluation based on these two issues, we have defined a fitness function having the following form:

$$E_{fit} = E_{net\_gen} + \lambda_{GA}(1 - E_{net\_size})$$

The first term (E_{net_gen}) accounts for generalization while the second term accounts for the network size. λ_{GA} is a weighting factor which controls the importance of the two terms. If λ_{GA} is very small, the fitness of a member is determined by its generalization performance only. However, when λ_{GA} is large, both generalization and size influence the fitness of a member. The value of the weighting factor λ_{GA} is determined adaptively, in a similar way λ_{WE} is determined in weight elimination. Specifically, λ_{GA} is determined as follows:

$$\lambda_{GA} = \lambda_{0\_GA} e^{-\beta_{GA}(1-E_{net\_gen})}$$

where λ_{0_GA} and β_{GA} are constants to be defined.

It is clear from the definition of the fitness function that reproduction favors members with good generalization performance and small network size. In early generations, network size does not play an important role in reproduction and the fittest members are the members which generalize best. However, in future generations both network size and generalization affect reproduction. For an estimation of E_{net_gen}, cross-validation is used again. Recalling our discussion in section 2, E_{net_gen} will be the generalization performance over the validation set, that is, $E_{net\_gen} = G_{val}$. E_{net_size} is defined to be the number of existing connections (flag=1) in a network belonging to a current population over the total number of connections in the first population. Both E_{net_gen} and E_{net_size} take values between 0 and 1.

4. Simulations and results

In order to evaluate our approach, a number of preliminary experiments has been performed using an artificial and a real database. For each

problem, data was divided into a training, a validation, and a test set. Three approaches have been compared: original back-propagation, weight elimination and genetic algorithm coupled with weight elimination. In all the simulations performed, the learning rate and momentum were both chosen equal to 0.1. For each problem, we chose an initial architecture (two-layer) and trained 20 different networks with different initial weights. The population size P in the genetic algorithm approach was also set equal to 20. The generation gap was set equal to 1.5 at the first population and its value was allowed to increase linearly during successive generations and reach the value 1. The β_{GA} was set equal to 40, while γ was set equal to 0.9 in all the simulations. λ_{0_WE} and λ_{0_GA} were both set to 1. The same set of initial weights, β_{WE} and w_0 was used in all the approaches to ensure comparable results.

TABLE 1: Results on the Numbers database

Method	%train				%test				weights			
	avg.	sd	best	worst	avg.	sd	best	worst	avg.	sd	best	worst
Back-propag.	1.0	0.0	1.0	1.0	0.82	0.01	0.85	0.81	2230	0.0	2230	2230
Back-propag. and weight elim.	1.0	0.0	1.0	1.0	0.82	0.02	0.84	0.79	1450	368.8	1174	649
Genetic alg. and weight elim.	1.0	0.0	1.0	1.0	0.85	0.004	0.87	0.84	1184	3.0	1185	1188

4.1. Numbers database

This is an artificial database which consists of noisy versions of machine printed numbers, digitized in a 7x9 grid. There are totally 10 classes. The training set consists of 150 examples, the validation set of 50 examples and the test set of 150 examples. The architecture chosen for this experiment was a fully connected two-layer network with 63 nodes in the input layer, 30 nodes in the hidden layer and 10 nodes in the output layer. The total number of weights and biases for this architecture is 2230. Table 1 illustrates the average, best, and worst performance in terms of generalization on the training and test sets, as well as in terms of network size.

TABLE 2: Results on the Ionosphere database

Method	%train				%test				weights			
	avg.	sd	best	worst	avg.	sd	best	worst	avg.	sd	best	worst
Back-propag.	1.0	0.0	1.0	1.0	0.94	0.005	0.94	0.925	1112	0.0	1112	1112
Back-propag. and weight elim.	0.96	0.01	0.97	0.95	0.94	0.01	0.95	0.91	184	65	156	84
Genetic alg. and weight elim.	0.98	0.0	0.98	0.98	0.97	0.0	0.97	0.97	149	0.2	143	153

4.2. Ionosphere database

This database consists of radar data. It contains 2 classes and a total of 351 instances. The number of attributes is 34, all of which are real numbers in the interval [0,1]. The database is distributed into two different files, a training file including 200 instances and a testing file including 151 instances. In order to create a validation set also, we split the training file into two different files. The first consisted of 200 examples and was our actual training set, and the second consisted of 31 examples and was our validation set. The architecture chosen for this experiment was a fully connected, two-layer network with 34 nodes in the input layer, 30 nodes in the hidden layer and 2 nodes in the output layer. The total number of weights and biases for this architecture is 1112. Table 2 illustrates the average, best, and worst performance in terms of generalization on the training and test sets, as well as in terms of network size.

5. Conclusions

In this paper, a new approach was introduced where a genetic algorithm was coupled with weight elimination. The purpose was to improve generalization by reducing network size. Preliminary results indicate the feasibility of this approach. However, we are currently performing a large number of experiments, using many real databases, in order to further validate our

approach.

References

[1] D. Hush and B. Horne, "Progress in supervised neural networks", *IEEE Signal Processing Magazine*, pp. 8-39, Jan. 1993.

[2] G. Bebis and M. Georgiopoulos, "Feed-forward neural networks: why network size is so important", *IEEE Potentials*, pp. 27-31, October/November 1994.

[3] E. Karnin, "A simple procedure for pruning back-propagation trained neural networks", *IEEE Transactions on Neural Networks*, vol. 1, no. 2, pp. 239-242, 1990.

[4] M. Mozer and P. Smolensky, "Skeletonization: a technique for trimming the fat from a network via relevance assessment", in *Advances in Neural Information Processing Systems 1*, pp. 105-115, 1989.

[5] Y. Chauvin, "A back-propagation algorithm with optimal use of hidden units", in *Advances in Neural Information Processing Systems 1*, pp. 519-526, 1989.

[6] C. Ji, R. Snapp, and D. Psaltis, "Generalizing smoothing constraints from discrete samples", *Neural Computation*, vol. 2, pp. 188-197, 1990.

[7] A. Weigend, D. Rumelhart and B. Huberman, "Generalization by weight elimination with application to forecasting", in *Advances in Neural Information Processing Systems 3*, pp. 875-882, 1991.

[8] S. Fahlman and C. Lebiere, "The Cascade-Correlation learning architecture", in *Advances in Neural Information Processing Systems 2*, pp. 524-532, 1990.

[9] M. Frean, "The Upstart algorithm: a method for constructing and training feed-forward networks", *Neural Computation*, vol. 2, pp. 198-209, 1990.

[10] Y. Le Cun, B. Boser, J. Denker, D. Henderson. R. Howard. W. Hubbard, and L. Jackel, "Back-propagation applied to handwritten zip code recognition", *Neural Computation*, vol. 1, pp. 541-551, 1989.

[11] S. Nolwan and G. Hinton, "Simplifying neural networks by soft weight sharing", *Neural Computation*, vol. 4, no. 4, pp. 473-493, 1992.

[12] H. Thodberg, "Improving generalization of neural networks through pruning", *International Journal of Neural Systems*, vol. 1, no. 4, pp. 317-326, 1991.

[13] Y. Hirose, K. Yamashita, and S. Hijiya, "Back-propagation algorithm which varies the number of hidden units" *Neural Networks*, vol. 4, pp. 61-66, 1991.

[14] J. Sietsma and R. Dow, "Creating artificial neural networks that generalize", *Neural Networks*, vol. 4, pp. 67-79, 1991.

[15] Y. Yang and V. Honavar, "Experiments with the cascade-correlation algorithm", Technical Report, Dept. of Computer Science, Iowa State University, Ames, IA.

[16] R. Kamimura and S. Nakanishi, "Weight-decay as a process of redundancy reduction", in *Proceedings of World Congress on Neural Networks*, vol. III, pp. 486-489, 1994.

[17] D. Goldberg, *Genetic algorithms in search, optimization, and machine learning*, Addison-Wesley, Reading, MA., 1989.

[18] X. Yao, "A review of evolutionary artificial neural networks", appeared in the *International Journal of Intelligent Systems*.

[19] K. Hornik and M. Stinchombe, "Multilayer feed-forward networks are universal approximators", in f2Artificial Neural Networks: Approximation and Learning Theory, H. White et. al., Eds., Blackwell press, Oxford, 1992.

[20] G. Miller, P. Todd and S. Hegde, "Designing Neural Networks using Genetic Algorithms", *Third International Conference on Genetic Algorithms*, pp. 379-384.

[21] D. Montana and L. Davis, "Training feed-forward neural networks using genetic algorithms", in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 762-767, 1989.