An Approach to Encode Multilayer Perceptrons

Jerzy Korczak and Emmanuel Blindauer

Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection, CNRS, Université Louis Pasteur, 7, Bld S. Brant, 67400 Illkirch, France.
{jjk,blindauer}@lsiit.u-strasbg.fr

Abstract. Genetic connectionism is based on the integration of evolution and neural network learning within one system. An overview of the Multilayer Perceptron encoding schemes is presented. A new approach is shown and tested on various case studies. The proposed genetic search not only optimizes the network topology but shortens the training time. There is no doubt that genetic algorithms can be used to solve efficiently the problem of network optimization considering not only static aspects of network architecture but also dynamic ones.

1 Introduction

One of the major domains of application on genetic algorithms (GA) is searching in a large space for good solutions and the optimization of hard problems [1], [4], [6], [14]. Compared to other methods, GA is perfectly capable to explore discontinued spaces of solutions. The space exploration is guided by a fitness function, and enables to evolve the population in parallel, rather than focusing on a single best solution. Among many types of network models, Multilayer Perceptron networks are the most common, not only because of their universality but also because of their good performance [12], [17]. It is known that the efficiency of learning and the quality of generalization are strongly related to the neural network topology. The number of neurons, their organization in layers, as well as their connection scheme have a considerable influence on network learning, and on the capacity for generalization [5], [18].

2 Network Representation and Encoding Schemes

The most common practice in neural network representation is to store only a network topology, but one may also encode the initialization parameters, connection weights, transfer functions, learning coefficients, etc. The more parameters are encoded, the bigger is the search space. In consequence, taking too many parameters will at least slow down the convergence of the genetic process. This excess cost will decrease the quality of solutions, especially if the inserted parameters are irrelevant.

The main goal of an encoding scheme is to represent neural networks in a population as a collection of chromosomes. There are many approaches to

J.R. Dorronsoro (Ed.): ICANN 2002, LNCS 2415, pp. 302–307, 2002.

genetic representation of neural networks [3], [8], [9], [10], [13], [16]. A perfect network coding scheme does not exist, because of a number of requirements that should be fulfilled by a good encoding scheme. A detailed discussion of all such requirements may be found in [2], [16].

To introduce the problem of encoding, let us consider one of most common methods used to encode networks. The direct encoding scheme consists of mapping the network structure onto a binary connection matrix which determines whether a connection between two neurons exists or not. The complete network structure is represented by the list of neurons with their incoming connections

Besides the direct encoding scheme many other interesting methods exist. Several authors proposed an encoding scheme based on grammar rules. Kitano [10] developed a method founded on rewriting rules. A more topology-oriented method is proposed by Korczak and Dizdarevic [3], [11].

Classical methods for encoding neural network use to encode the network topology into a single string [3], [8], [9], [10], [13], [16]. But frequently, for large-size problems, these methods do not generate satisfactory results, in terms of learning performance and simplicity. We have observed in the experiments that computing new weights to get satisfactory networks is very costly. Even using fast computing methods did not help a lot because thousands of back propagations had to be done on large datasets. After evaluating all networks, crossover and mutations, the population is composed of new individuals, but none of them have inherited from the weights previously computed. So a lot of computing time can be lost here.

Our idea is to keep the networks in the learned state and to transmit the knowledge acquired to their children. In several cases, choosing bad initial weights causes the learning procedure converges to a local minimum and not the global. In consequence, the network is usually considered not efficient. Therefore in the genetic process, we should reduce the fact that some networks may disappear from the potential search space, due to bad starting points.

Encoding a phenotype in a string is also an essential point in the evolution. Operators, such as mutation or crossover, are very dependent on the genotype. It should be ensuring that splitting a string implies an interesting cut in the network. Taking into consideration the above mentionned problems, a new encoding method based on the matrix encoding is proposed.

Let n be a maximal number of neurons who can be used in all networks. The network can be represented by a matrix $M \in M_{p,p}(\mathbb{R})$ where p is the number of neurons in the current network. $M_{i,j}$ equals to zero if there is no connection between the neuron i and the neuron j, else $M_{i,j}$ represents the weight of the connection.

The matrix is symmetrical, thus it is only needed to store the right superiors. Layers in the neural network are visible in the matrix as blocks of zeros over the diagonal. Using this encoding scheme, any network can be represented in a unique way assumed that neurons are sorted by weight within a layer. The matrix representation can be used as the genotype of the network. This encoding scheme respects all properties proposed by [8], [16] for evaluating the quality of an encoding: completeness, closure, proximity, short schemata, compactness, non

isomorphism and modularity. These properties do not claim to be complete, and more criteria may be proposed. But these properties can be considered as the minimal set of requirements for a genetic encoding to be efficient.

Two operators for a genotype have been proposed: a crossover operator, and a mutation operator. For the crossover operation, a new matrix $M^{1,0}$ is created from two other matrices $M^{0,0}$ and $M^{0,1}$. $M^{1,0}$ is created by extracting l_1 columns from $M^{0,0}$ and $p - l_1$ columns from $M^{0,1}$. In result, the network is splitted vertically into two parts, and the offspring get two different parts, one from each parent. A more generalized crossover could be an exchange of a sub-matrix between the parents. For the mutation, a few random values in the matrix are changed. This operation can create or remove several connections, depending on the mutation coefficient. So are weights modified during the mutation. Thus, all solutions in the search space are available by the genetic algorithm.

3 Weight Initialization and Updating

With this algorithm, the weights are included within the genetic encoding. Thus, after the learning stage, trained networks are available, and the weights are reused in the next population. In the evolution process, after crossover and mutation, resulting networks have weights inherited from their parents. It can be expected that the knowlegde issued from the parents would help to speed up the convergence of the gradient based algorithms.

Gradient-based methods look for the best gradient in the surface of error. It should be pointed out that the weight adapting only occurs when weight is changed. When no connection exists between two neurons, current methods based on gradient do not create the connection, because the search space from the weights $\mathbb{R}^p$ is included in the space of all weight available $\mathbb{R}^n$. Thus, we have the inclusion $\mathbb{R}^p \subset \mathbb{R}^n$. The computed gradient is done in the $\mathbb{R}^p$ space, and thus it indicates the best slope. But nothing can guarantee that in the $\mathbb{R}^n$ space does not exist another vector with a better slope: when the slope of the gradiant in $\mathbb{R}^p$ is close to zero, it is considered to be close to a minimum. But, computing the gradiant in $\mathbb{R}^n$, we could get another vector indicating the best slope allowing to converge faster to the minimum. Below, an algorithm to find the best slope and to determine qualitativly the importance of connections between neurons is proposed.

The connection weights are computed in the following way:

Let $\bigcup_{i=1}^{q} \bigcup_{k=1}^{s} \{(X_i, S_k(X_i))\}$ be the set of data, where q is the number of elements, S_k is the projector onto the k-th element, and s the number of outputs. $R_k(\omega, X_i)$ is the k-th output of the network. So the total quadratic error, on all of the dataset, is computed according to the formula:

$$E = \sum_{i=1}^{q} \sum_{k=1}^{s} [R_k(\omega, X_i) - S_k(X_i)]^2$$

thus

$$\frac{\partial E}{\partial \omega} = 2. \sum_{i=1}^{q} \sum_{k=1}^{s} \left[R_k \left(\omega, X_i \right) - S_k \left(X_i \right) \right] \frac{\partial R}{\partial \omega} \left(\omega, X_i \right)$$

To compute $\frac{\partial R}{\partial \omega}$, the Jacobian matrix from E is computed. The term $\frac{\partial R}{\partial \omega} (\omega, X_i)$ can be evaluated as:

- if only one path drives from ω to the output, then $\frac{\partial R}{\partial \omega} (\omega, X_i) = z. \prod_j (\omega_j . f'(z_j))$, where z is the incoming value in the connection, $\prod \omega_j . f'(z_j)$ the product of all weight and the derived number from the the activation function on the path to R.
- if several paths drive from ω to the output, we just have to add each computed value for each path, as previously.

Example: A four neurons network is presented in Fig.1, where f is the activation function, x and y the input values, and A, B, C, D are values after the activation of respective neurons, a, b, ... m are the weights. R is the output value.

The values of $\frac{\partial R}{\partial a}$, and $\frac{\partial R}{\partial c}$ are computed as follows:

- $\frac{\partial R}{\partial a} = xf'(ax + by).e.f' \left(e.A + m.D \right) .g.f' \left(g. \left(f \left(e.A + m.D \right) + h.D \right) \right)$
  $= x.A'.e.B'.g.C'$
- $\frac{\partial R}{\partial c} = x.D' \left(m.B'.g.C' + h.C' \right)$

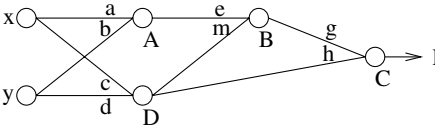

Fig. 1. Computing weights

To compute weights, all dataset is presented to the network, and all partial derivatives are added to construct the Jacobian. Each element in the matrix indicates if a connection is important or not, a large number indicates that the connection has a large impact on the output. Contrary to the normal gradient method, in our algorithm, only one step has to be applied for changing weights. Note that the method changes the topology of the network, because new connections may be created or deleted. But constraints may be defined in order to keep the network topology unchanged.
Using the Jacobian, relevance of the inputs can be take into account. Small values indicates an irrelevant entry. In the same way, we use the Jacobian to

eliminate connection between neurons: with a small value in the Jacobian and a small weight, we can take out the connection and perhaps the neuron, when it has no more input or output.

4 Experimentation

To evaluate the performance of the new encoding scheme, we used several classical problems: `xor`, `parity 3,4,5,` `Heart`, and `Sonar`. These case studies have been chosen based on the growing complexity of the problem to solve. Each population had 200 individuals. For each individual, 100 epochs were carried out for training. An maximal number of hidden neurons is fixed for each problem. 20 runs were done for each problem, only 4 runs were carried out for `heart`, because too much time was needed. In the genetic part, the crossover percent is set to 80%, and 5% for the mutation (the mutation rate was decreasing with the number of generation).

The learning was stopped when the best individual reached a test error under 5%, or the number of generation exceeded 500 iterations (in this case, the last test error is written in brackets). In the experiments, the fitness function was a non-linear combination of the learning error, the test error, the topology of the network, and of the number of generation. Compared with other results

Table 1. Results of experimentations

	XOR	Parity 3	Parity 4	Parity 5	Heart	Sonar
Number of hidden neurons	2	3	5	8	12	30
Number of connections	6	11	23	38	354	1182
Number of epochs (error)	13	23	80	244	209 (9%)	120 (13%)

from [7], [15], this new method has shown the best, not only in term of network complexity, but also in quality of learning (for the `Heart` problem, the error is only 9%, compared to other methods with at least 15%, and for the `Sonar` problem, 13% against 30%).

5 Conclusion

In this paper, application of genetic algorithms for the topological optimization of Multilayer Perceptron is presented. This approach is more flexible to the design of different topologies than heuristic approaches.

The new encoding scheme improves the efficiency of evolutionary process, because the weights of the neural network have been included in the genetic encoding scheme. The experiments have confirmed that firstly by encoding the network topology and weights the search space is affined; secondly, by the inheritence of connection weights, the learning stage is speeded up considerably. The

presented method generates efficient networks in a shorter time compared to actual methods. The evolution process improved the quality of network populations gradually and discovered near-optimal solutions. The genetic connnectionism approach has been shown as a robust optimization method with a large spectrum of applications. Of course, the scope of GA is restricted to those problems only where it is possible to encode the set of networks as chromosomes, and where a fitness function may be defined.

References

1. R.F. Albrecht, C. R. Reeves and N. C. Steele (eds), *Artificial Neural Nets and Genetic Algorithms*, Springer Verlag, 1993.
2. E. Blindauer, *Méthodes d'encodage génétique de réseaux neuronaux*, MSc thesis in Computer Science, Louis Pasteur University, Strasbourg, 1998.
3. E. Dizdarevic, *Optimisation génétique de réseaux*, MSc thesis in Computer Science, Louis Pasteur University, Strasbourg, 1995.
4. L. Eshelman, Proc. of the Sixth Intern. Conf. on *Genetic Algorithms and Their Applications*, Morgan Kaufmann, 1995.
5. D.B. Fogel, L.J. Fogel and V.M. Porto, Evolving Neural Networks, *Biological Cybernetics* 63: 487–493, 1990.
6. D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning,* Addison-Wesley, 1989.
7. M.A. Grönroos, *Evolutionary Design Neural Networks*, PhD thesis, Department of Mathematical Sciences, University of Turku, 1998.
8. F. Gruau, *Neural Networks Synthesis using Cellular Encoding and the Genetic Algorithm*, PhD thesis, LIP, Ecole Normale Superieure, Lyon, 1992.
9. F. Gruau, Genetic Synthesis of Modular Neural Networks, [in] S. Forrest, (eds), *Genetic Algorithms: Proc. of the 5th International Conference*, M. Kaufman, 1993.
10. H. Kitano, Designing Neural Networks using Genetic Algorithms with Graph Generation System, *Complex Systems*, 4: 461–476, 1990.
11. J. Korczak and E. Dizdarevic, *Genetic Search for Optimal Neural Networks*, [in] Summer School on Neural Network Applications to Signal Processing, Kule, 1997.
12. Y. Le Cun, *Modeles Connexionnistes de l'Apprentissage*. PhD thesis, Paris, 1987.
13. M. Mandischer, Representation and Evolution of Neural Networks, [in] R.F. Albrecht, C.R. Reeves and U.C. Steele (eds), *Artificial Neural Nets and Genetic Algorithms*, pp. 643–649, 1993.
14. Z. Michalewicz, *Genetic Algorithms+Data Structures=Evolution Programs*, 3rd ed. Springer, Berlin, 1996.
15. J.C.F. Pujol, R. Poli, *Evolving the Architecture and Weights of Neural Networks Using a Weight Mapping Approach*, School of Computer Science, Birgmingham, 1999.
16. R. Salustowicz, *A Genetic Algorithm for the Topological Optimization of Neural Networks*, Diplomarbeit TU Berlin, 1995.
17. D.M. Skapura, *Building Neural Networks*, Addison-Wesley, 1996.
18. X. Yao, *Evolving Artificial Neural Networks*, Proc. of the IEEE, 87(9):1423–1447, 1999.