Evolutionary Design of MLP Neural Network Architectures

Elson Felix Mendes Filho, André Carlos Ponce de Leon Ferreira de Carvalho
Computational Intelligence Laboratory
Computer Science and Statistics Department - ICMSC - USP
Av. Carlos Botelho, 1465 - São Carlos, SP - Brazil
{prico, andre}@icmsc.sc.usp.br

Abstract

In Neural Networks design, some parameters must be adequately set for an efficient performance to be achieved. Such parameters include: learning rate, momentum term, number of hidden layers, number of units in each hidden layer, etc. The setting of these parameters is not a trivial task since different applications may require different values. The "trial-and-error" or traditional engineering approaches for this task do not guarantee that an optimal set of parameters is found. Besides, the choice of the best parameters depends on the particular problem to be solved and the statistical distribution of the datasets. Evolutionary approaches have been recently proposed to overcome these problems. Genetic Algorithms have been used as a heuristic search technique to find adequate neural architectures. This technique, first developed by Holland in 1975, is based on natural selection and genetic mechanisms. It tries to produce better individuals from an initial set of individuals, based on a given criteria. This paper presents some results achieved by using this technique to search optimal neural architectures to solve real world credit analysis problems. In this paper, the searches have been restricted to Multilayer Perceptron, MLP, networks. The MLP networks used in this work are feed-forward fully-connected strictly-layered architectures.

Keywords: Genetic Algorithms, Neural Networks, Finance.

1. Introduction

The performance of Multilayer Perceptron [15] Neural Networks strongly depends on the topology of these networks (size, structure and connections). It is also influenced by the parameters of its learning algorithm (learning rate, momentum term). As a result, the definition of the network architecture (topology and parameters) has a large effect on its performance, characterizing its learning process speed, learning precision, noise tolerance and generalization capacity.

It is usually very difficult to project efficient neural networks. There are several techniques, in which some empirical knowledge is involved, that can help in special cases [6]. A very common approach is the design of networks structures from standard architectures or based on another previously used architecture. In these cases, the network is tested for the desired function. Afterwords its architecture is modified until a suitable architecture is found [1].

This approach implies a long design time, may present a very high cost and does not guarantee the best results. This happens because the performance criterion is based on a complex arrangement of factors. This is a typical problem of "multi-criterial" optimization. Aiming to solve this problem, several automation techniques to design neural architectures for particular classes of problems have been investigated. One of these techniques involves the use of an evolutionary approach like Genetic Algorithms as a search heuristic to find approximately optimum architectures.

Genetic Algorithms [10] provide a more natural approach for the solution of neural design, especially because the human brain is also, somehow, a result of the biological evolution. According to Gerald Edelman [7], Nobel Prize in 1972, the human mind is a direct result of natural selection. He suggests that the organization of neurons and their connections were not pre-determined, but were evolved to compete with the circumstances imposed by the environment.

The brain architecture was, during millions of evolution years, astutely molded through genetics and natural selection. Thus, it is promissing to use an artificially equivalent approach to evolve artificial "beings", like Artificial Neural Networks.

0-8186-8070-9/97 $10.00 © 1997 IEEE

2. Evolutionary Design of Neural Architectures

In the evolutionary design of neural architectures, each individual can be seen as a state in the network space. Beginning the process with a population of genotypical representations of valid networks, randomly generated, a generator reconstructs these networks (phenotypes) from its representation (genotypes), according to the transformation function used. As a result, a simulator trains all the networks with the same training dataset, and evaluates their performances are evaluated.

Thus, some data are used to evaluate the present state of the population, establishing the fitness, of its individual. The fitness is an aptitude grade for each network, which measures its performance regarding all other networks.

Through a method similar to a roulette wheel, the networks are selected according to a relative probability associated with their fitness. This is performed until the pool of candidates is filled; the candidates will be set to reproduction guided by mutation and cross-over genetic operators; through these operators a new generation of neural architectures is constructed. In order to avoid the best networks quickly disappearance from the population, an elitist policy can be used, which cause these networks to be present in the next generation.

This cycle is repeated and, with the passing of new generations, the population evolves gradually towards genotypes which correspond to phenotypes with higher performances. This is carried out a certain number of times until the algorithm finds appropriate solutions to the problem, or, in extreme cases, until it arrives to the best solution. During this process, the best networks, as well as some statistical data, can be collected and stored for a posterior analysis. Figure 1 illustrates this process.

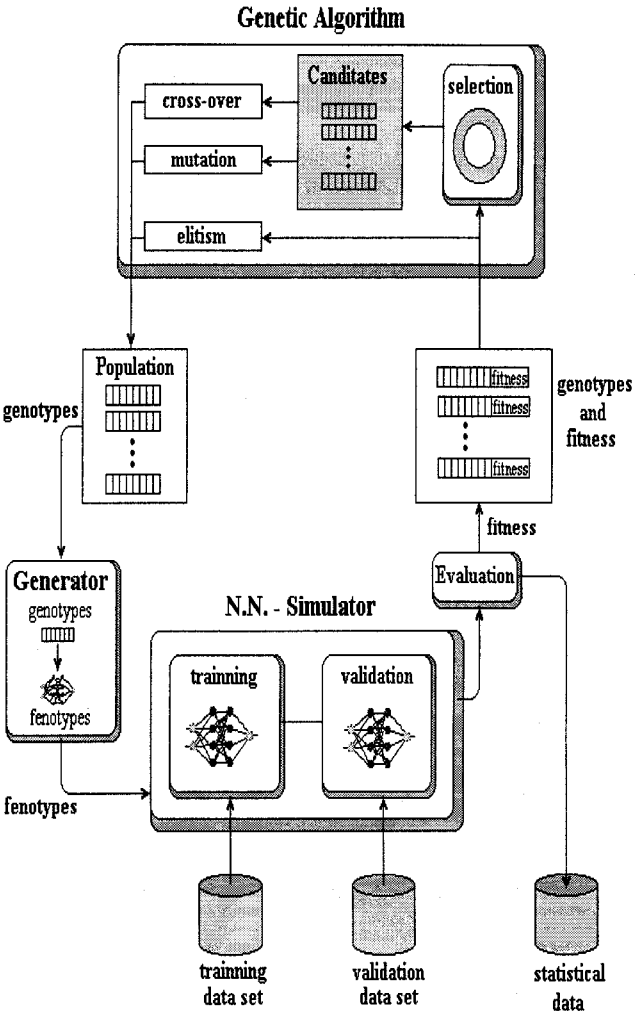

Figure 1. Evolutionary approach process for Neural Network architecture design.

2.1. Representation Approaches

The problem of how a neural architecture is genotypically (基因型) represented is crucial in the design of an evolutionary system. The representation or codification used determines not only the classes of neural architectures which could evolve, but also the functioning of both the decoding process and the reproduction operators [2].

The representation of a neural network structure is not straight. Several aspects should be considered: if the representation allows approximately optimum solutions to be represented; how invalid structures could be excluded; how the reproduction operators should act so that the new generation would have only valid Neural Networks; and how the representation would support the growth of neural architectures. The ideal situation would be that where the genetic space of the neural networks do not have genotypes of invalid networks. It is also desirable that this space could be expanded to all genotypes of potentially useful networks [3].

There are two main paradigms of Neural Network representation: the direct or low-level representation, and the indirect or high-level representation. There is a clear dis-

tinction between these two [16].

A direct representation specifies exactly each parameter of the network, and requires little effort in decoding, because the transformation of genotypes into phenotypes is direct. An example is the coding in a matrix of connections, which specifies in a direct and precise way the connections of a Neural Network. A current method of direct representation for the genotypical coding of a neural topology is the mapping of the structures in the form of binary connection matrices, where each matrix element determines whether a connection between two units exists or not, as shown in the method presented in [12]. This method is illustrated in Figure 2.

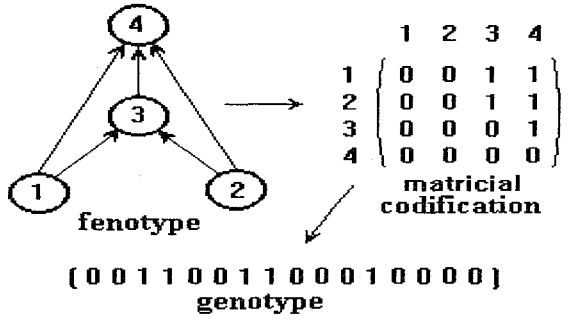

Figure 2. Example of direct representation.

This method is obviously aimed at the choice of connections, since the other parameters of the neural topology must be previously established. The main problem of this approach is that incorrect structures can be produced. It can, for example, produce feedback connections for feedforward networks. Another problem of this approach is the necessity of very large codes for large networks. This method is useful only for relatively small network topologies. One advantage of this approach is that it can be easily modified in order to become a training algorithm or an initial weights choice algorithm, including in the matrix the weights values of the connections.

Indirect representations, on the other hand, require a considerable effort for the decoding in the construction of the phenotypes. However, this kind of representation can use abstract descriptions or grammatical coding to characterize the networks. Besides, the networks can be pre-structured, using restrictions in order to exclude undesirable architectures, which makes the searching space much smaller.

A few sophisticated methods work with indirect representations, using a description of the network through several parameters. These parameters may be the number of layers, size of the layers and the connections among them. This representation also allows the inclusion of restrictions in the networks to reduce the number of incorrect structures. As a consequence, the number of structures to be trained and evaluated, as well as the number of evolutionary cycles necessary to produce good networks, are drastically reduced.

One of the most popular methods, presented by Mandisher [11], involves basically an improvement of the technique presented by Harp [9]. This method is aimed at the choice of the architecture and connections, and uses a representation which describes the main components of the networks. These components are divided in two areas: parameter area and layer area.

In this method, the parameter area specifies the learning rate and the momentum term for all the network connections. Each layer has its own layer area, specifying the number of units on the layer and the connections to other layers. The connections are classified as projective (connections with posterior layers) and receptive (connections with anterior layers), and they are specified through parameters of density and radius. The radius parameter specifies the radius of connections for each unit, and is given by the percentage relative to the size of the destination layer. The density parameter specifies the density of the connection of a receptive unit, and is given by the percentage of units which are connected to this unit. This representation is shown in Figure 3.

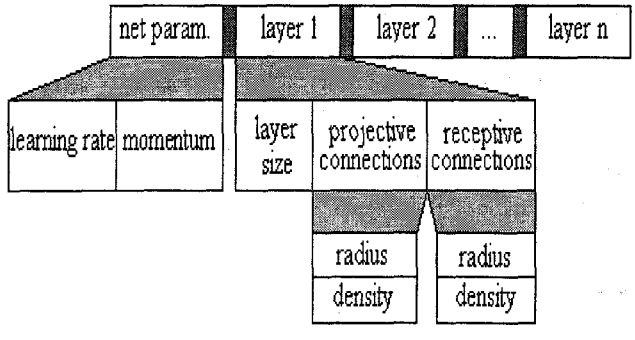

Figure 3. Example of indirect representation.

This work is concerned only with the choice of the architecture. The connections will be determined by the training algorithm, making the Genetic Algorithm search space considerably smaller. Thus, the networks can be completely connected and strictly layered at the beginning. The Backpropagation training algorithm will establish the weights of the connections and these could be pruned through a prun-

ing algorithm when they are shown to be of little importance.

Therefore, and on account of the available computational resources, it was chosen to create a representation that did not concern itself with the connections and was more direct. It is essentially a simplification of the representation proposed in [11].

This representation also describes the components in parameter area and layer area. The parameter area specifies the learning rate and the momentum term for all network connections. The layer area specifies the number of units in each layer (up to three layers were used). As it is illustrated in Figure 4, this representation requires a smaller decoding effort, on account of its simplicity. Differently from [11] and [9] representations, the decoding is direct and precise, that is, the decoding generates one architecture only.

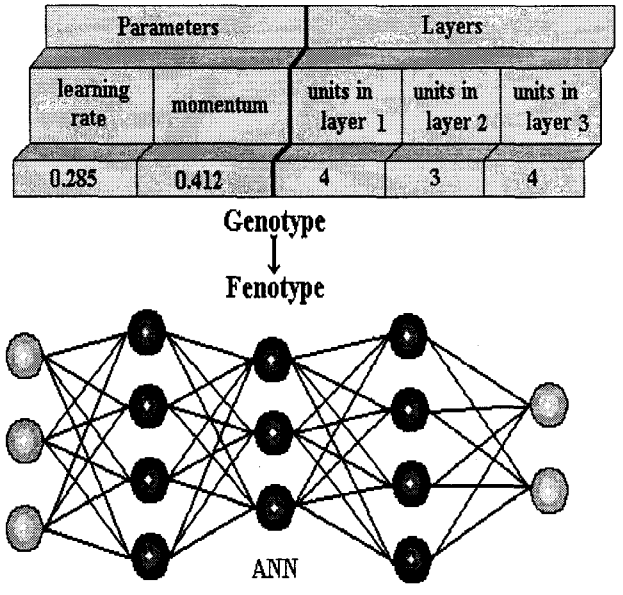

Figure 4. Proposed representation example.

In this representation it is necessary to specify only the number of input and output units, beyond the maximum number of units in the hidden layers. Invalid structures could be generated in only one case: when there were not any units in the second hidden layer, although there were units in the third hidden layer. However, this case is easily detectable, and it is not generated in the initialization phase.

2.2. Performance Evaluation

The networks can only be evaluated after they have been trained using the same dataset. The training and validation results can be used to determinate the performance of these networks. If there is availability, a second test set can be used to determinate the performance of these networks with data not previously seen. This would be useful as a validation for the Genetic Algorithm.

In order to create a function which associates a fitness value tor each network, different heuristics can be used. These heuristics should take important aspects into account such as: error in the training and validation sets, training time and network size. Such heuristics should weigh the importance of these aspects, depending on the behavior desired for the application.

For some applications, such as credit evaluation, to which the created system is used, it is possible to specify heuristics of performance evaluation that took into account the error costs made by the network, through a cost matrix, or the calculation of losses caused by the network in each transaction. Thus, the evaluations can be more specific and precise, and the given fitness value can be more representative of the network performance. In the credit evaluate problem the fitness can be calculated as shown by Equation (1).

$$Fitness = MaxCost - (TCost + 2 * VCost) \quad (1)$$

Where:

*MaxCost = (MaxTCost + 2*MaxVCost)*
*MaxTCost = (C1*NTPC1 + C2*NTPC2)*
*MaxVCost = (C1*NVPC1 + C2*NVPC2)*
*TCost = (C1*NTPWC1 + C2*NTPWC2)*
*VCost = (C1*NVPWC1 + C2*NVPWC2)*
C1 = Cost of wrong classification of patterns of the class 1
C2 = Cost of wrong classification of patterns of the class 2
NTPCn = Number of training patterns of the class n
NVPCn = Number of validation patterns of the class n
NTPWCn = Number of wrongly classified training patterns of the class n
NVPWCn = Number of wrongly classified validation patterns of the class n

As the evaluation of neural architectures comprises training, the computational cost is very high. However, considering the inherently parallel characteristics of the algorithms involved, its utilization in distributed environments or its parallel implementations can reduce this cost.

2.3. Reproduction

The reproduction strategy is the most important aspect of a Genetic Algorithm. The definition of this strategy must take into account the characteristics of the representation to be used, the necessities of the problem and the available computational resources. In the definition of the reproduction strategy, the mechanisms of selection, the elitist policy, the cross-over and mutation operators with their respective occurrence rates must be specified. These aspects will be analyzed in the following subsections.

2.3.1 Selection Mecanism

In this work the classical roulette wheel method was used to select the candidates. It is based on relative fitness, once the goal is the choice of the best architectures, while keeping the diversity of the population. A variant of this method, based on ranking, favors individuals with best fitness, and, thus, may reduce the diversity in excess. However, this variant can operate better than the classical method when the fitness values are very close [17].

Through this method, networks with higher fitness are preferentially selected. These networks are put in a pool of candidates that could be manipulated by the genetic operators.

2.3.2 Elitism

In order to avoid the disappearance of the architecture with the highest fitness from the population when this populations is manipulated by the genetic operators, an elitist policy was used. By using this policy, the architecture with the highest fitness is automatically put in the next generation. It may happen that many architectures have the same highest fitness. In this case the chosen play-off criterion is the dimension of the topology, favoring, in this way, the network with the smallest number of units.

2.3.3 Genetic Operators

The genetic operators are used to transform the population of neural architectures through the generations, diversifying this population and keeping the desirable acquired characteristics. In this work these characteristics were: learning rate, momentum term and the number of units in the hidden layers.

The cross-over genetic operator, the predominant operator, is responsible for the recombination of network characteristics during the reproduction, allowing the next generations to inherit desirable characteristics.

This operator can make an exchange of characteristics between two architectures, which are selected with a given probability by the cross-over rate Pc. By means of a randomly chosen number a cross-over point is specified in which characteristics of these architectures will be exchanged, as illustrated in Figure 5.

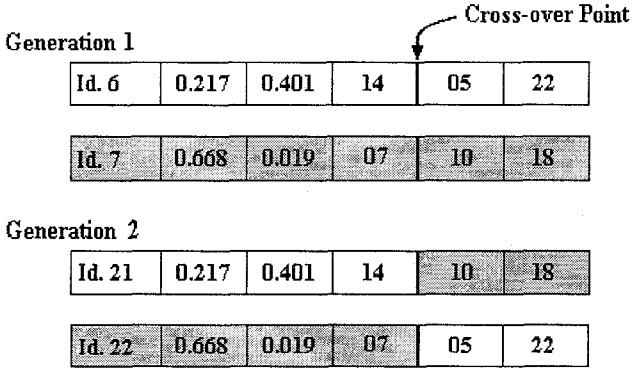

Figure 5. Example of the proposed cross-over operator.

In the specification of this operator, there is always concern about the validity of the generated architectures. Using the proposed representation, there is only one case in which an invalid architecture could be generated: when one of the architectures presents zero units on the second and third hidden layers, and the cross-over point is defined between these two layers. In this case, the operator will generate a network with zero units in the second hidden layer and N units in the third hidden layer. However, this special case is easily identified and treated, so, only correct network architectures participate in the evolutionary process.

The action of the mutation genetic operation, the secondary operator, must cause only slight qualitative changes in the characteristics of the architectures during the reproduction, allowing the next generations to diversify themselves.

One architecture is chosen for mutation with probability given by the mutation rate Pm, which is smaller than the cross-over rate. Thus, through a randomly chosen number, the point of mutation is specified in which an architecture characteristic is changed to an also randomly chosen value, following maximum and minimum limits defined for each characteristic, as illustrated in Figure 6.

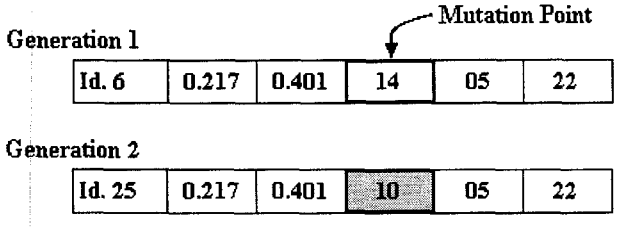

Figure 6. Example of the proposed mutation operator.

In the specification of this operator, there is also a concern with the validity of the generated architectures. As stated for the cross-over operator, using the proposed representation there is only one case in which an invalid architecture is generated: when the architecture presents zero units on the second and third hidden layers, and the chosen mutation point is related to the number of neurons on the third layer. This special case, whose occurrence is even more rare than the occurrence of the cross-over special case, is also easily identified and treated.

3. NeurEvol

During the development of this work, a system prototype for the evolutionary design of neural architectures was created, the NeurEvol. This system was implemented using the C language. The created prototype consists of two main modules: the Evolutionary Module and the Neural Module.

The Evolutionary Module consists of a Genetic Algorithm and it has evaluation and reproduction sub-modules. The Neural Module works by exchanging files with the SNNS simulator [18] for the Neural Network training and testing processes. This module consists of a network generator sub-module, a batchman program generator sub-module and a sub-module for reading the results produced by the simulator.

This prototype operates in the following way: initially, a population of neural architectures representations is randomly generated. These representations are sent, one by one, to the Neural Module which generates network files in the SNNS simulator format with the specifications of these networks and the batchman program files for training and test. These programs are processed using the simulator, which makes the training and test of these networks and generates files with the results. These files of results are read and the performance data is sent to the sub-module of evaluation, which gives the relative fitness for each representation. The representations and their fitness are sent to the reproduction sub-module, which uses the selection mechanisms and the genetic operators for the creation of a new generation of the population which is evaluated again. This process is repeated for a specified maximum number of generations.

At the beginning of this process, the number of individuals in the population, the maximum number of generations, the cost functions for the performance evaluation, the cross-over and mutation rates, the datasets to be used, the maximum number of units in the hidden layers, the number of training cycles and the stop criterion for the training can be specified by the user, according to the necessities of the application and the resources available.

3.1. Experiment

In these experiments using the NeurEvol, a dataset of real world credit card applications from UCI machine learning databases repository was employed [13].

The NeurEvol was used with a cross-over rate of 80% and mutation rate of 10%. During the network training the smallest value of the Mean Squared Error in the training and validation sets was used as a stop criterion. Besides, a N-cross-validation scheme with three datasets was used. Thus, each architecture was trained and tested three times using three different permutations of the dataset and its fitness reflected the average of the three results achieved.

The NeurEvol was used with a population of 20 individuals during 10 generations. It was specified the maximum number of 20 units in the hidden layers, and the training lasted for 800 cycles.

Figure 7 presents the fitness of the best individual while Figure 8 presents the average fitness of the population in each generation. It can be observed that the fitness of the best individual improved in the generations 4, 8 and 10; whereas the average fitness was evolved gradually. These results also show that several good architectures were obtained in few generations.

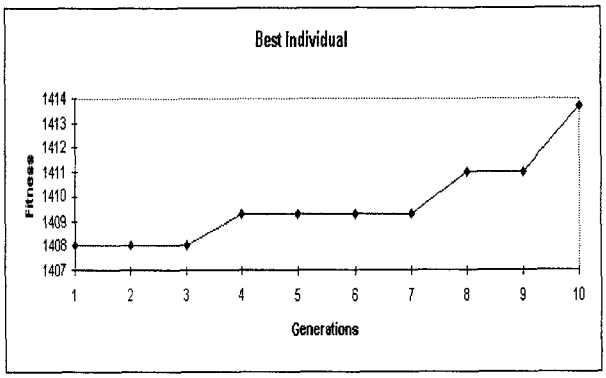

Figure 7. Evolution of the best individual during 10 generations.

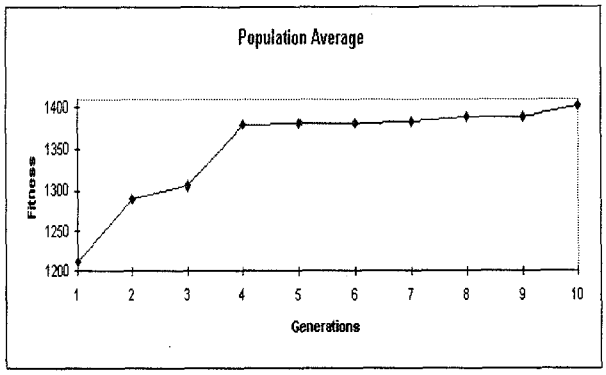

Figure 8. Evolution of the average population during 10 generations.

The results obtained in this experiment were compared to previous results presented in [5], using the methods ID3 and C4. Table 1 presents the average number of patters wrongly classified in each class in the test set (ANPWC) and the costs of these errors using these methods.

4. Conclusions and Future Work

The optimization of Neural Networks using Genetic Algorithms is usually carried out using toy problems. In this paper, the optimization for a real world application was investigated. In the experiments, restrictions related to the application were considered in the optimization process. These restrictions were useful to reduce the search space used by the Genetic Algorithm.

Table 1. Results of the experiment x previous results

Method	*ANPWC - C1*	*ANPWC - C2*	*Cost*
ANN - NeurEvol	19	18	73
ID3	7	28	91
C4	7	19	64

The results presented in this paper show that is possible to automate the design of neural architectures using an evolutionary approach. It was shown that this approach is able to improve the set of network parameters towards an optimal architecture. Besides, the improvement in the average performance of the networks population makes it possible to construct a system which integrates several networks in a network committee.

The results presented in this paper, concerning the use of evolutionary Neural Networks for credit assessment, may provide an effective tool to support the work carried out by managers and credit analists, allowing a better control, cost reduction and improving customers assistance in large scale credit operations.

This work was concerned only with the optimization of Multilayer Perceptrons networks trained with the Backpropagation algorithm. However, the method proposed is general enough to be used with other connectionist models or different training algorithms. The system is already being modified to work with Multilayer Perceptrons networks trained by QuickProp [8] and RProp [14], as well as Radial Basis Function models [4].

For future work, an increase in the number of generations, the removal of the restrictions of fully connected and strictly layered networks, the use of alternative fitness functions, the investigation of other representation approaches and the analysis of the performance achieved by NeurEvol for other datasets are planned.

References

[1] D. Bailey and D. Thompson. Developing neural network applications. *AI Expert*, 5(9):34–41, September 1990.

[2] K. Balakrishnan and V. Honavar. Evolutionary design of neural architectures - a preliminary taxonomy and guide to literature. Technical report, A. I. Research Group, Iowa State University, 1995.

[3] J. Branke. Evolutionary algorithms for network design an training. Technical report, Institute AIFB, University of Karlsruhe, 1995.
[4] D. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2, 1988.
[5] C. Carter and J. Catlett. Assessing credit card applications using machine learning. *IEEE EXPERT*, Fall 1987.
[6] M. Caudill. Neural network training tips and techniques. *AI Expert*, 6(1):56–61, January 1991.
[7] G. Edelman. *Neural Darwinism*. Basic Books, New York, 1988.
[8] S. Falman. *Faster-learning variations on back-propagation: An empirical study*. Morgan Kauffman, 1988.
[9] S. Harp, T. Samad, and A. Guha. Towards the genetic syntesis of neural networks. In *4th International Conference on Genetic Algorithms*, pages 360–369. Morgan Kaufmann, 1991.
[10] J. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975.
[11] M. Mandisher. Representation and evolution of neural network. Technical report, University of Dortmund, Germany, 1993.
[12] G. Miller, P. Todd, and S. Hedge. Designing neural networks using genetic algorithms. In *3rd International Conference on Genetic Algorithms*, pages 379–384. Morgan Kaufmann, 1989.
[13] C. Murphy and D. Aha. UCI repository of machine learning databases. Technical report, Irvine CA, University of California, 1994.
[14] M. Riedmiller. Rprop - description and implementation details. Technical report, Universitat Karlsruhe, 1994.
[15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Parallel Distributed Processing*, volume 1: Foundations, chapter Learning internal representations by error propagation, pages 318–362. The MIT Press, 1986.
[16] G. Weib. Combining neural and evolutionary learning: Aspects and approaches. Technical report, Technische Universitt Mnchen, 1990.
[17] D. Whitley. The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *3rd International Conference on Genetic Algorithms*, pages 116–121. Morgan Kaufmann, 1989.
[18] A. Zell et al. Snns stuttgart neural network simulator - user manual, version 4.1. Technical report, I.P.V.R., Universitt Stuttgart, Germany, 1995.