An Evolutionary Method to Find Good Building-Blocks for Architectures of Artificial Neural Networks

Christoph M. Friedrich
University of Witten/Herdecke
Inst. for Technology Development and Systems Analysis
Alfred-Herrhausen Str. 44
58455 Witten, Germany
chris@uni-wh.de

Claudio Moraga
University of Dortmund
Department of Computer Science
44221 Dortmund, Germany
moraga@ls1.informatik.uni-dortmund.de

Abstract

This paper deals with the combination of Evolutionary Algorithms and Artificial Neural Networks (ANN). A new method is presented, to find good building-blocks for architectures of Artificial Neural Networks. The method is based on *Cellular Encoding*, a representation scheme by F. Gruau, and on Genetic Programming by J. Koza. First it will be shown that a modified Cellular Encoding technique is able to find good architectures even for non-boolean networks. With the help of a graph-database and a new graph-rewriting method, it is secondly possible to build architectures from modular structures. The information about building-blocks for architectures is obtained by statistically analyzing the data in the graph-database. Simulation results for two real-world problems are given.

1 Introduction

One of the major problems using ANN's is the design of their architecture. The architecture of an ANN greatly influences its performance. If the architecture is too small, the net is not able to learn the desired input/output mapping. On the other hand, if the architecture is too large, the net generalizes poorly on unseen data. Some constructive and pruning algorithms have been suggested. A constructive algorithm starts with a small network and grows it. Pruning algorithms start with a big network and remove nodes or connections until the network error increases. None of these algorithms are able to find good performing architectures for a great set of problems. The constructive algorithms make restrictive assumptions for the building-blocks of the architecture, so not all architectures can be build. Pruning algorithms involve the problem of choosing the initial architecture and the decision which part of the network should be removed. Some pruning algorithms like OBS [6] are very computing-intensive. As a conclusion these algorithms find only solutions for special sets of problems. Many users find the architecture in a time-consuming trial and error cycle. Miller, Todd and Hedge [10] call it a *black art* and suggest that Evolutionary Algorithms (EA's [2]) are good candidates for an automated search in the space of possible network architectures because the fitness function associated with such a space has several properties which make it unappealing for classic methods: it is complex, noisy, non-differentiable, multimodal and deceptive. Much work has been done in this area. For a survey of recent work the paper of Branke [3] is suggested. Most of the difficulties in this area arise from the problem of choosing the right representation, to encode a network graph for the Evolutionary Algorithm.

*Appeared in: Proceedings of the Sixth International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU '96); Pages 951-956; Granada, Spain (1996).

2 Cellular Encoding

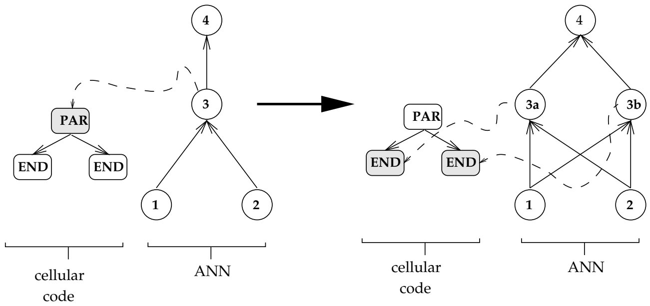

Figure 1: A sample graph-rewriting step in the development of an ANN using Cellular Encoding

A method to optimize the architecture and weights of boolean neural networks, where the weights are restricted to the values -1 and 1, was suggested by F. Gruau [5] who named it Cellular Encoding. In this encoding technique, the information to develop an architecture is obtained by interpreting the information from a grammar-tree. The nodes of this tree encode information about graph rewriting operations. The development of a neural network starts with a graph consisting of one node (cell), having ingoing connections from the input area and outgoing connections to the output area. Every cell has a reading head pointing to one node of the grammar-tree (cellular code). The nodes of the cellular code contain symbols defining the graph rewriting operation. The operation #par: symbolizes a parallel cell division, both cells inherit the input and output connections from the mother cell. The reading heads of the new created cells are moved to the left and right subtree of the grammar-tree. Figure 1 shows a sample step in this development proccess. For biological plausibility, the development of the cells should be parallel. This can only be achieved by using a FIFO Queue. The development of a network ends, if all cells evaluate the #end: operation, located as terminal at the leaves of the grammar-tree.

As mentioned above, the cellular code is given as a grammar-tree. In this case it is possible to use Genetic Programming [8] to optimize the cellular code. Genetic Programming is a Genetic Algorithm, that uses trees instead of Bitstrings as representation. The recombination operator crossover is realized by exchanging the subtrees of two parent individuals. The mutation operator brings some variations to the genome and is realized by inclusion of randomly initialized trees.

Gruau has proved some properties of his encoding technique, among them are completeness, compactness, closure, modularity and scalability.

3 Modified Cellular Encoding

For the optimization of the architecture of ANN's with real-valued weights, the Cellular Encoding method was modified. Some new Development operators were developed and tested. The set of operators {#par:, #seq:, #seqcopy:, #usplit:, #lsplit:, #end: } was used to develop the architectures. Figure 2 shows the effects of rewriting the hidden node of a network with the development operators. The operation #par: makes a parallel splitting of a cell, whereas #seq: makes a sequential one. Different to Gruau's work, it is possible with the operator #seqcopy:, to create networks with shortcut connections. This is important because it is claimed by many scientists that shortcut connections can lead to better performing networks [10]. The

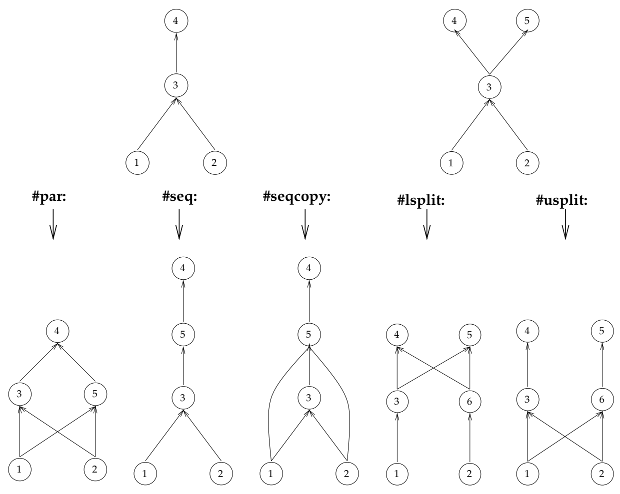

Figure 2: The used rewriting operations for the modified Cellular Encoding

new introduced operations #usplit: and #lsplit: make parallel splittings, but seperate the ingoing or outgoing links of the created nodes. This makes it possible to create sparsely connected networks with hierachical architectures. One property of the created networks should be their correctness. A created graph is a correct architecture for a feedforward neural network, if it contains only feedforward connections and if there are no isolated nodes, which means that all hidden nodes are on a path from an input node to an output node. The operators #par:, #seq: and #seqcopy: guarantee this property. The operators #usplit: and #lsplit: can create incorrect network graphs in the case of a single in- or outgoing connection. To avoid this the operators were defined to copy the middle connection to both offsprings, if the fan-in or fan-out is odd. Together with the closure under the recombination operator crossover, it is only possible to create correct feedforward network architectures during the evolutionary cycle.

The second modification to the evolutionary development cycle was the use of a graph-database. This database contains the cellular code, network graphs and fitness parameters like the number of necessary epochs for learning, classification error on the learning and test set and other parameters of the networks. Figure 3 shows the working structure of the evolutionary cycle. Before the fitness calculation of a developed network (by using the network simulator SNNS 4.0 [14]), it is checked, whether the fitness of this network was calculated five times before. If not, the fitness is calculated again and saved in the database. The resulting fitness of a network is the mean of all fitness evaluations of this network. This method allows it to minimize the fitness distortion resulting from the ran-

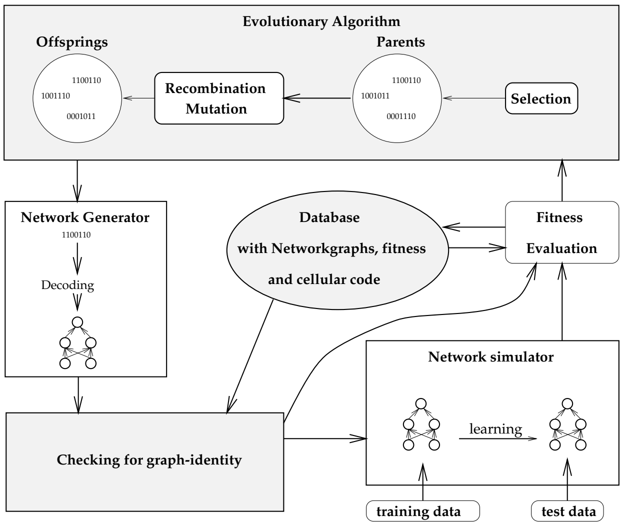

Figure 3: Modified Cellular Encoding using a graph-database

dom initialization of the weights. This increases the validity of the obtained results. Other authors document only solutions which are results of a single fitness evaluation. In this work, only results are given for architectures tested five times. Another advantage of the database is the possibility to save computing-expensive fitness evaluations, if the network was tested five times before. One problem is the testing if a network graph is included in the database. The best possible way of testing would be, checking if two graphs are isomorph. Two not-identical graphs are isomorph, if they are structurally identical, that means if it is possible to find a permutation of the node labels that make the graphs identical. Unfortunately the problem of testing graph isomorphism for the set of correct feedforward network graphs with shortcut connections is NP-complete [7]. The problem of graph isomorphism for the optimization of neural network architectures has been known before by many scientists who call it: problem of the *competing conventions*. It leads to bigger search spaces, because of the genotype-phenotype mapping of Genetic Algorithms. It is possible with some genotypic representation techniques to have many genotypes, that code functional equivalent (isomorph) networks. Unfortunately a crossover between functional equivalent network representations can lead to offsprings which have inferior fitness in comparison to their parents. In this work the test for inclusion in the database was done by checking for graph identity. Using a special graph-labelling algorithm, with some heuristics, it was possible to label many isomorph graphs identically. This minimizes the effects of the problem of isomorph graphs.

3.1 Genetic Engineering

Optimizing the architecture of an ANN using Evolutionary Algorithms leads to a proper performing architecture for a problem. But it gives no insight into the problem, how good architectures are built. It would be much more interesting to find building principles for good architectures. Especially for Genetic Programming, where the representation of a problem is a grammar-tree, Altenberg [1] suggested a method to find good building-blocks that are subtrees of the representation. He called this technique *Genetic Engineering*. It depends on the assumption, that some parts of the genetic program have a higher impact on the complete fitness of the genetic program than others. It is a great problem to attach a fitness value to subprograms, because it is only possible to obtain the fitness of the complete phenotype. In this work the fitness of building-blocks of genetic programs was found by analyzing the cellular code in the graph-database, which was build during the formerly described optimization process. The fitness for a building-block was found by counting the occurences of the subtree in the cellular codes of the database. For this simplification the assumption was made, that during the evolutionary process the number of good performing architectures will increase and with it the occurence of good building-blocks. These building-blocks are called modules.

3.2 Graph-Rewriting

Having found good building-blocks for neural network architectures, it would be interesting to see how architectures built of this modules perform. In this work a special graph-rewriting method was used. The (i, k)-rewriting of a network graph with a module starts with a fully connected $n - i - m$ network, where n, m are determined by the problem. Then all hidden nodes are rewritten k times with the rewriting operations defined by the module. For $k = 0$ this results in standard architectures with one hidden layer. With this method it is possible to create families of architectures consisting of good performing building-blocks. Figure 4 shows an example of a network created by $(1, 1)$-rewriting of a 3-1-2 network with the module usplit:par:lsplit:. The hidden node will be first rewritten with a #usplit: operation and the resulting nodes will be rewritten with the operation #par: or #lsplit:.

4 Experimental Results

The data for the experiments are taken from the benchmark suite from L. Prechelt [11]. The task for the diabetes problem is to classify from some dia-

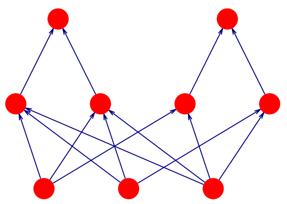

Figure 4: Resulting network of $(1,1)$-rewriting of a 3-1-2 network with the module usplit:par:lsplit:

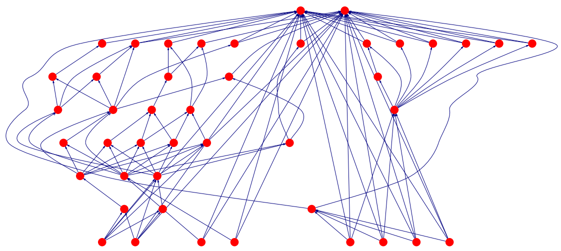

Figure 5: optimal architecture for the diabetes problem, a net with 44 nodes and 110 weights

gnostic data (e.g. blood pressure, result of glucose tolerance test etc.), whether a female Pima indian was diabetes positive or not. It is a classification problem with 576 learning patterns and 192 test patterns. There are 8 input parameters and 2 output classes. The classification method is WTA (winner takes all). Prechelt and Michie [9] found networks having a classification error of 24.8 % on the test set. The activation function of the network was the $tanh(x)$. Riedmiller [12] suggested, that this symmetric function (a scaled and shifted logistic function) can lead to faster learning than the asymmetric standard sigmoid. The learning set was presented 500 times, as learning algorithm, RPROP [13] was chosen. This adaptive learning algorithm learns about 5 times faster than standard backpropagation and needs less computing time than backpropagation (Prechelt also uses RPROP in his experiments). Unfortunately the network simulator supports no cross-validation, thus learning was done without it. The population size for the Genetic Algorithm was 50. The fitness function for this problem was a linear combination of the classification error on the test set (factor 1) and the learning set (factor 0.3). Several optimization runs with this problem were made. About 20,000 architectures were tested, this task needed 12 days of SparcStation 10 computing time. Using the graph-database saved about 5 % of the fitness evaluations. Figure 5 shows the best architecture found, it is a network with 44 nodes (34 hidden nodes) and 110 weights. This sparsely connected network shows a classification error of 23.125 % on the test set in five runs. It can be seen that the nodes of the architecture are distributed over several layers.

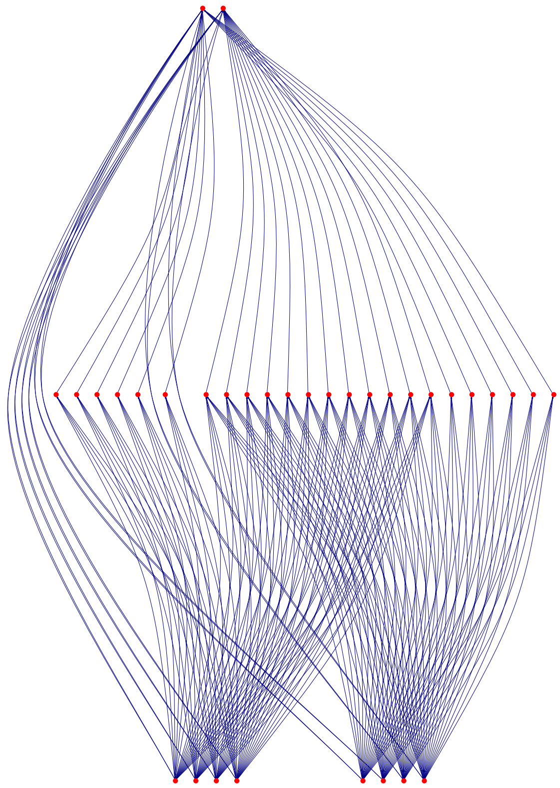

Figure 6: architecture for the diabetes problem created by a (6,1)-rewriting with the module usplit:par:lsplit a network with 34 nodes and 184 weights

The application of Genetic Engineering on the data in the graph-database found the module usplit:par:lsplit:. This module will make a #usplit: rewriting with a node and then a #par: or #lsplit: operation with the child nodes. Compare Figure 4 for the effects of this rewriting on a 3-1-2 network. The elements of the family of architectures that can be constructed through $(i,1)$-rewriting perform very well on this problem. The architecture created by $(6,1)$-rewriting (shown in Figure 6) even performs better in the mean of five runs than the best net found by the Evolutionary Algorithm. This net with 34 nodes (24 hidden nodes) and 184 weights shows the same classification error of 23.125 % on the test set, but the classification error on the learning set was about 10 % better than that of the optimized architecture. The hidden nodes of this network are only on one hidden layer, but it can be seen that the architecture shows a hierarchical structure.

The second tested problem was the cancer-problem, also from the benchmark suite of Prechelt. The task in this problem is to classify a tumor of the female

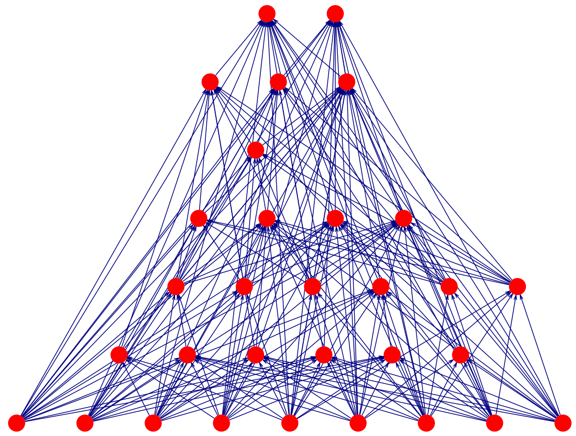

Figure 7: optimal architecture for the cancer problem, a net with 31 nodes and 202 weights

breast either benign or malignant. The input parameters are obtained by microscopic examination of tumor cells. The problem with 9 input parameters and two output classes was learned with a set containing 525 elements. The test set to quantify generalization ability had 174 elements. The fitness function was a linear combination of classification error on the test set (factor 1) and learning set (factor 0.3). Furthermore, the number of nodes (factor 0.02) and the number of used epochs (factor 0.04) determined the searched architecture. All other parameters were set as in the diabetes problem. 100 generations with a population size of 50 were tested. This results in 5000 fitness evaluations. For this evaluations 12 hours of SparcStation 10 computing time were needed. The best found architecture, a net with 31 nodes (20 hidden nodes) and 202 weights is shown in figure 7. The fitness for this network was 5.26 in the mean of five runs. Standard architectures, tested up to 32 hidden nodes show a fitness around 7. Prechelt [11] obtained similar results for standard architectures, but it must be noted, that he uses cross-validation, so the number of learning epochs are not directly comparable. He found for his cancer3 dataset an optimized architecture showing a fitness of 5.8, this architecture had 35 nodes and 436 weights build up in a 9-16-8-2 structure. As a conclusion the evolutionary optimized architecture is the best found architecture, because for a similar fitness it uses half of the weights than the architecture of Prechelt. Furthermore it should be noticed that the structure of the best found network is strictly hierarchical.

The application of Genetic Engineering found the module seqcopy:par:par:. The family of architectures build by $(i, 1)$-rewriting shows a fitness in the mean of five runs of 5.5 for $i \in \{2-8\}$. Even more complex rewritings like the $(2, 2)$-rewriting lead to an architecture with fitness 5.67. Figure 8 shows the graph

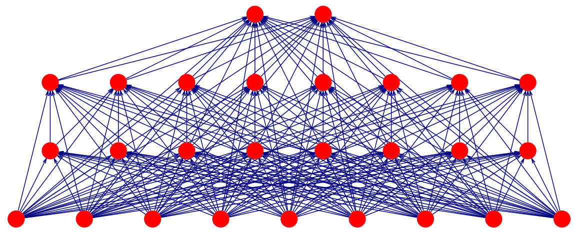

Figure 8: architecture for the cancer problem created by $(4, 1)$-rewriting with the module seqcopy:par:par:

obtained by $(4, 1)$-rewriting with this module. This network with 27 nodes and 194 weights shows a strict hierarchical structure.

5 Conclusion

It was proved that it is possible to find good performing architectures for non-boolean ANN's, with a modified Cellular Encoding method. The functionality of the method was demonstrated on two real world problems. The results were compared with results available from literature. All evolved architectures show better performance for the optimized criteria than comparable standard architectures. With the help of a graph-database it was furthermore possible to find building-blocks for the architecture of ANN's. Architectures created with a new graph-rewriting method from this modular structures show better performance than standard architectures, in one problem they performed even better than the optimized architecture found by the Evolutionary Algorithm. Using this technique, it is possible to create a library of good building-blocks for problem-specific network architectures. Further investigations on these building-blocks may give some new insights into the working principles of ANN's.

References

[1] L. Altenberg. The evolution of evolvability in genetic programming. In K. E. Kinnear, editor, *Advances in Genetic Programming*. MIT Press, 1994. ftp://ftp.mhpcc.edu/pub/incoming/altenberg/LeeEEGP.ps.Z.

[2] T. Bäck. *Evolutionary Algorithms in Theorie and Practice*. PhD thesis, University of Dortmund, 1994.

[3] J. Branke. Evolutionary algorithms for neural network design and training. In *Proceedings of the 1st Nordic Workshop on Genetic Algorithms and its Applications*, 1995. ftp://ftp.aifb.uni-karlsruhe.de/pub/jbr/Vaasa.ps.

[4] C. M. Friedrich. Entwicklung und Analyse Evolutionärer Algorithmen zur Optimierung der Struktur und Parameter von künstlichen Neuronalen Netzen. Master's thesis, University of Dortmund; Department of Computer Science; 44221 Dortmund; Germany, 1995.

[5] F. Gruau. Cellular encoding of genetic neural networks. Technical Report 92-21, Ecole Normale Superieure de Lyon, Institut IMAG, 1992. ftp://lip.ens-lyon.fr/pub/Rapports/RR/RR92/-RR92-21.ps.Z.

[6] B. Hassibi and D. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In S. Hansen, J. Cowan, and C. Giles, editors, *Advances in Neural Information Processing Systems (NIPS) 5*, pages 164–171. Morgan Kaufmann, 1993.

[7] C. M. Hoffmann. *Group-Theoretic Algorithms and Graph Isomorphism.* Springer, 1979. LNCS 136.

[8] J. R. Koza. *Genetic programming, on the programming of Computers by means of natural selection.* MIT Press, 1992.

[9] D. Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine Learning, neural and statistical classification.* Ellis Horwood Ltd., 1994.

[10] G. F. Miller, P. M. Todd, and S. U. Hegde. Designing neural networks using genetic algorithms. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 379–384. Morgan Kaufmann, 1989.

[11] Lutz Prechelt. PROBEN1 — A set of benchmarks and benchmarking rules for neural network training algorithms. Technical Report 21/94, Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany, September 1994. ftp://ftp.ira.uka.de/pub/papers/techreports/1994/1994-21.ps.Z.

[12] M. Riedmiller. Schnelle adaptive Lernverfahren für mehrschichtige Feedforward-Netzwerke, Vergleich und Weiterentwicklung. Master's thesis, Universität Karlsruhe, Fakultät für Informatik, 1992. ftp://ftp.ira.uka.de/pub/neuron/papers/-Adaptives_Lernen.ps.gz.

[13] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In H. Ruspini, editor, *Proceedings of the IEEE International Conference on Neural Networks (ICNN)*, pages 586–591, 1993. ftp://i11s16.ira.uka.de/pub/neuro/papers/riedml.-icnn93.ps.Z.

[14] A. Zell et al. *SNNS User Manual, Version 4.0.* University of Stuttgart, Institute for Parallel and Distributed High Performance Systems, Breitwiesenstr. 20-22; 70565 Stuttgart, 4.0 edition, 1995. ftp://ftp.informatik.uni-stuttgart.de/pub/SNNS.