Evolving Feedforward Neural Networks – an outcome of simulation studies

Halina Kwasnicka [*]

Wroclaw University of Technology, Institute of Applied Informatics, Wroclaw, Poland, email: halina.kwasnicka@pwr.wroc.pl

Abstract. The paper is designed as a summary of the relatively wide study on usage of Evolutionary Algorithms (EAs) as an automatic tool for Neural Networks (NNs) design. The study was conducted for many years and partial results have been presented. Special attention is put to such aspects of NNs evolution as chromosome coding, defining fitness function, and different evolutionary approaches. The studied approaches are shortly described and only general results, summing up all experiments are presented. Summary contains some recommendations formulated on the basis of experiments. Future works are also mentioned.

1 Introduction

Genetic Algorithms (GAs) and artificial Neural Networks (NNs) were proposed as the imitation of natural proceses: biological evolution and real neural systems. Designing architecture of neural networks causes some problems: How many neurons should be in the network? How they should be organized? How many connections we should establish, and between which neurons? A designer of a NN architecture must relay on his experience or on informal heuristics. He must select an adequate architecture between huge number of possible networks. This task can be perceived as a searching process of a large space of potential solutions.

The papers describing combination of the two above mentioned techniques started to appear in the late 1980's [17, 15, 14, 16, 2, 5, 19]. GAs can be used for optimization:

- numbers of layers, neurons and/or connections,
- connection weights (instead of usage a training algorithm),
- values of adjustable parameters associated with a model of NNs,
- whole structure of a NN.

The process of NN architectures design is still rather a matter of art, not the routinized task. A designer of NNs must rely on his experience and on the informal heuristics arising from the works of others researchers [17]. We can distinguish two types of non-evolutionary approaches – *constructive* and *destructive*. Features of the domain of possible architectures cause these methods not to give satisfying results [19]. The space

[*]Acknowledgement. This work was financed from the State Committee for Scientific Research (KBN) resources in 2006 - 2007 years as a research project No 3 T11C 039 30.

of possible architectures are infinitely large, nondifferentiable, complex, noisy, deceptive and multimodal.

The paper discusses a problem connected with usage a GA as a NN designing tool. This problem has been studied by the author for years, some results have been described in earlier papers (i.e., [7, 8, 10, 11, 12, 13]):

- Efficiency of NN architecture designing using direct and indirect coding schema. It is important issue because a used coding schema limits a space of potential solutions for a GA.
- Efficiency of NNs architecture designing using different fitness functions. A fitness function determines selection pressure and therefore influences the direction of evolution.

In this paper we will try to find some general advices for evolving NNs. The paper is structured as follow. The next section introduces into direct and indirect coding schemata. Section third presents usage a coding schema with pleiotropy and polygene effects. Section fourth describes an approach proposed and tested by Ferreira [3, 4]. The results are discussed in Section fifth. The last section contains short resume.

2 Direct and indirect coding schemata

A **direct encoding** stores a NN as a directed graph. Genes are directly converted to elements of the phenotype – neurons and connections between them. Such description gives the largest possibilities in NN architectures generation. However, this approach charcterizes relatively high memory complexity, equal to $O(n^2)$, n is a number of neurons.

A genotype in direct encoding consists of two elements: vector V and matrix E. V contains information about nodes, E – information about connections. Both, V and E contain only binary values. Such genotypes allow not only for representation of feedforward NNs, but also recurent ones. Genetic operators operate separatelly on vector V and matrix E (E is treated as a binary vector of n^2 length.

An **indirect coding** characterizes more complex decoding process, each chromosome (individual) must be processed into the NN. The search space is strongly limited – only a subset of NN archtectures can be represented in the form of grammar sentences. A chromosome (binary tree) is coded as a list of symbols from some predefined grammar.

Our grammar consists of three symbols:
S – represents sequential division of the processed neuron (PN). The created neuron (CN) receives all outputs of the PN. The PN's output is connected to the CN's input.
P – represents parallel division of the PN. The CN is an exact copy of the PN, it contais the same input and output connections. There is no connection between the PN and the CN.
E – represents a terminal symbol. The PN is not divided anymore.

All leaves of the tree have to contain the terminal symbol E. The information located in the leaves is invariant and do not need to be represented. The number of neurons in the hidden layer is equal to the number of S and P symbols in a genotype plus 1. The approach requires specialized genetic operators, working on the lists of symbols.

Both, direct and indirect coding schemata alow to use traditional or evolutionary training method. When we use evolutionary training, we must add a list of weights into a chromosome.

3 A coding with pleiotropy and polygenic effects – GAPP

Usually GAs use a simple relation: one gene – one phene, what means that each gene is directly a coded parameter of an optimized function. In biology, a single gene of individual can have an impact, at the same time, on several phenotype features. Such an effect is called a *pleiotropy.* On the other hand, each single phenotype's feature of an individual (its phene) can be determined by simultaneous influence of a number of genes – this effect is called a *polygene* [9]. One of the first attempts to use the pleiotropy and polygenic effects (but not for NNs design) is made by Altenberg [1], where the final fitness function is formed as a sum of a number of components. Sequentially, a new gene is included into a chromosome and components on which it influences are randomly selected. The problem with usage this approach to a NN architecture design is to separate independent phenotypic features of a NN as components.

In our approach, the genotype in GAPP is a vector of numbered genes g_i, $i = 1, ..., N$. Each gene is a binary coded real value, therefore a chromosome is a string of bits. Mutation and crossover are implemented as in the classic GA. The phenotype is a vector of numbered M phenes: $f_1, ..., f_M$, phenes are real values. The binary *pleiotropy matrix* (called PP) is used during decoding process, each element of PP is equal to 0 or 1.

$$[Genotype]_{1xN} \times [PP]_{NxM} = [Phenotype]_{1xM} \tag{1}$$

Binary PP can be easy modified, using mutation operator. In our experiments we have assumed: designed NNs are full-connected and feedforward, neurons have bias, numbers of input neurons (**noIn**), output neurons (**noOu**) and (maximal) hidden neurons (**noHi**) are known, sigmoid activation functions, real valued outputs (from *0* to *1*). Each chromosome is a list of weights (from bias to hidden neurons, from bias to output neurons, from input neurons to hidden neurons, from input neurons to output neurons, from hidden neurons to output neurons, from hidden neurons to hidden neurons). By assumed (**noHi**) we decide about maximal number of weights (in total) and about dimension M of PP. A number of weights (**NoWeights**) is equal to:

$$\begin{aligned} NoWeights &= noHi + noOu + noIn \cdot noHi + noIn \cdot noOu \\ &+ noHi \cdot noOu + 0.5 \cdot noHi \cdot (noHi - 1) \end{aligned} \tag{2}$$

In the proposed algorithm we have included modification of PP matrix (see [10] for details). Modification of a single column of PP influences a single weight of the NN. Changes inside a column are accepted only if they improve the NN. Special operations enabling addition and deletion of neurons are also proposed. Improvement operations for PP matrix lie in: adding, deleting, modification of connections; adding, deleting and modification of a neuron in a hidden layer; modification of a matrix row (strength of the pleiotropy effect), modification of randomly selected elements in the PP. There are three types of PP modifications:

1. During initialization process – a number of individuals are randomly selected and modifications of PP is applied.
2. In selected generations during evolution (Mod_t).
3. If all phenotypes in the population are similar but the genotypes are different.

4 Gene Expression *P*rogramming (GEP) in NNs design

A Genetic Programming (GP) proposed by Cramer and developed by Koza [6] uses different coding schema than the GA, but the same paradigm of evolutionary computation. Individuals are a kind of parse tree instead of linear representation. Evolving individuals in the form of trees, as in GP, causes problems with using standard genetic operators, however GP can produce robust solutions of different tasks. Gene Expression Programming (GEP) [3, 4] is a relatively new idea of evolving complex structures in a form of trees as linear chromosomes.

Gene Expression Programming (GEP) was proposed by Ferreira in 1999 [3]. As Fereira said, GEP "is the inevitable development of GAs and GP, incorporating both the idea of simple, linear chromosomes of fixed length used in GAs and the ramified structures of different sizes and shapes used in GP." ([3], p. 21). The crucial issue in GEP is, that proposed chromosomes are capable of representing any tree. The structure of chromosomes uses multiple genes, each gene encodes a sub-tree (i.e., small program). As it was mentioned, a chromosome in GEP is a linear, symbolic, fixed length string. Having fixed length they are able to encode trees of different sizes and shapes. Equation 3 is an example of a simple mathematical expression and the corresponding GEP chromosome.

$$
\begin{gathered}
ln(x \cdot z + z/5) \\
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
Ln & + & \cdot & / & x & z & z & 5
\end{array}
\end{gathered}
\tag{3}
$$

Using GEP for NN design we must encode into a chromosome all elements that completely define a NN. The network architecture is encoded into a head/tail domain. The head is build from activation functions of hidden and output neurons and terminal (it represents input neurons). Additionally, a genom contains so-called NN-genes encoding the weights (Dw domain) and the thresholds (Dt domain).

Let us consider a simple NN with two input nodes, two hidden nodes and one output node. A NN and its tree form are shown in Fig.1, the corrresponding chromosome is done by eq. (4), digits encodes the weights.

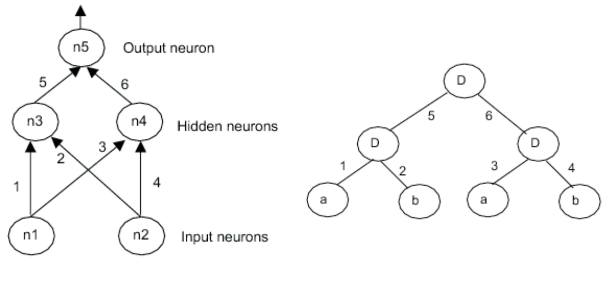

Figure 1. A NN and its representation, a and b denote the first and the second input respectively, D is a two-dimentional function.

The function D multiplies the values of the arguments by the weights assigned with them and adds all the incoming (weighted) values in order to determine the forwarded output.

$$\begin{array}{ccccccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 & 1 & 2 \\ D & D & D & a & b & a & b & 6 & 5 & 4 & 3 & 2 & 1 \end{array} \tag{4}$$

The weights are another structure, it contains all possible weights, but only required number of them are in use. The detailed description of the approach and possible genetic operators one can find in a number of Ferreira papers, e.g., [3].

5 The results

A relatively large number of experiments have been conducted to verify different evolutionary approaches. The popular test problems have been used in the experiments: XOR problem, Autoencoder, Thermometer, Evenness of a number, Addition of binary numbers (two bits), ZOO, HOUSING, IRIS, Generation of number sequences, Digits recognition, etc. Here we present selected results placing emphasis on important issues. The paper summarizes observations following from all experiments. We focus on the problems:

1. What kind of evolutionary algorithm should we use for NNs design?
2. Is the sophisticated approach better than a simple, classical GA?

Let us recall some results with relatively simple problem – XOR. The GA with indirect coding schema generates very efficiently, the proper, known from literature NN, but the GA with direct coding generetaes, a bit less effectively, but the optimal, compact NN (see Fig. 2a) and b)). Similarly, the GA with direct coding is able to evolve the optimal NN for evenness of numbers problem – see Fig. 3 (for this purpose only one input is important).

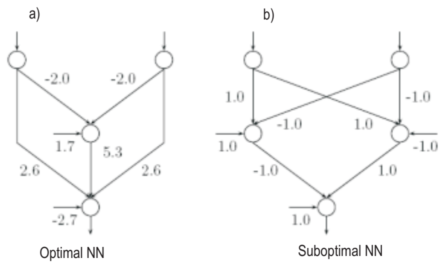

Figure 2. A NN for the XOR problem: a) optimal, direct coding, evolutionary learning, b) suboptimal, evolved with indirect encoding and evolutionary learning.

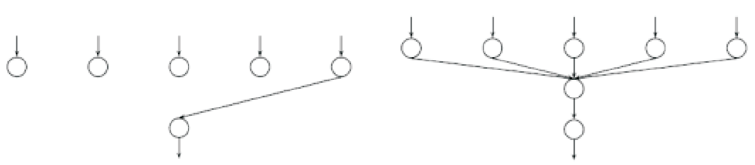

Figure 3. NNs for the evenness of numbers problem, evolved using direct and indirect coding and evolutionary learning.

Searching a NN for XOR problem and using GAPP, we have tested populations of 20 to 200 individuals. For $noHi = 1$, and $Tmax = 1000$, the classical GA (CGA) has problem with finding solution (only one or two successes for 20 runs) while **GAPP** gives good results. GAPP almost in all simulation runs has found the optimal NN in the assumed time (1000 generations), a number of successful runs varies from 90% to 100% (in contrast to 10% – 40% for a CGA). The high profit of usage of GAPP has been observed considering a number of needed genrations (50 – 100 GAPP and 800 – 1000 CGA). CGA works better (i.e., comparably with GAPP) if $noHi = 3$.

The frequency of a PP modification during evolution influences the tempo and mode of evolution. If modifications are made very often, the GA has no time for searching solutions' space. On the other side, rarely modifications cause that the population size starts play the role. The way of PP modification is the main problem with our method. Developing the more "intelligent" modification process would be very profitable.

Only two task have been studied with GEP approach: XOR and 6-bit Boolean multiplexer ([3, 4]). The experiments suggest that good results are obtained when GEP starts with short, single-gene chromosomes (compact organization) and then gradually increases a head (redundant organization). The best results are usually obtained with a moderately redundant organization. For XOR problem the results presented in [3] show that redundant system (RS) has success rate 77% but the compact system (CS) – 30%. An example of a proper, but redundant (too cmplicated) solution is presented by eq. (5,

$$\begin{array}{ccccccccccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ T & Q & b & a & b & a & a & b & b & 8 & 8 & 3 & 9 & 9 & 8 & 3 & 7 \end{array} \tag{5}$$

with weights' values [1.175, 0.315, -0.738, 1.694, -1.215, 1.956, -0.342, 1.088, -1.694, 1.288]. T represents a function of three arguments and Q represents a function of four arguments.

The small (compact) NN solving XOR problem is also found by GEP. The author of [3] (p. 142) underlines that "several other solutions to the XOR function were found that use exactly the same kind of structure of this parsimonious solution. Indeed, the algorithm discovered not only one but several Boolean functions of three arguments and invented new, unexpected solutions to the XOR problem. This clearly shows that gene expression programming is an astonishing invention machine, totally devoid of preconceptions."

It is worth mentioning that the success rate for this relatively simple problem is only 77% in redundant system and 39% in compact system. For the 6-bit Boolean multiplexer, the success rate varies from 4% (unigenic approach) to 6% (multigenic approach). Such results were obtained using a compact system – a head contained only one-, two- and three-dimensional functions.

GEP is a new and strongly developed idea. It seems that we should consider all above results rather as preliminary ones. Observing obtained success rates and taking into account that in both problems all examples were used as a training set, we can expect problems with more difficult tasks. This approach requires from a developer many decisions about assumed functions, number of genes, and used operators.

6 Summary

The main aim of the paper is to summarize the personal experience with usage evolutionary algorithms (EAs) as a NNs design tool. Results of study of different approaches

([7, 8, 10, 9, 11]) were exploited during deveoping the GANN system. In this section we focus on two questions:

1. What recommendations can be formulated on the basis of the experiments?
2. What further study emerges from conducted so far experiments?

Answering to the first questions, it is worth to underly that the experiments univocally have shown that efficiency of EAs very strongly depends on assumed coding schema and defined fitness function. Often a user focuses on genetic operators and other GA's parameters. We may spend a lot of time on tuning algorithm's parameters, developing specialized, sometimes sophisticated genetic operators. But the coding schema influences a size of search space. It can be very large and represent all possible solutions. Theoretically it is a very good situation – EAs can find the global optimum (see experiments with XOR and evenness problems, Fig. 2, 3). The word 'theoretically' in the previous sentence is used not accidentally. Finding the optimal solution in such a large, multimodal and deceptive search space is a very hard task. It requires long evolution, a GA can work unstable. Defining other coding schema, that delimits a search space can cause that satisfactory solution is found quicker and a GA works stable, its results are repeatable. Taking this into account, the first recommendation can be following:

If we are able to eliminate an unpromising part of search space by defining an adequate coding schema, the problem becomes easier. It is very important to assure that all promising solutions have to be representable in the assumed coding schema.

Defining fitness function we define a bias of our algorithm. For example, putting stress on the size of evolved NNs into a fitness function we can cause that our algorithm will have tendency to evolve small networks that are not able to learn the intended task. On the other hand, when a size of evolved network is not considered into a fitness function, the algorithm has tendency to generate relatively large NNs that learn the task by memorizing and have poor generalization ability. In the GANN system we have tested three different fitness functions trying to balance a NN's size and its generalization error. Those experiments allow for formulation the second recommendation:

A fitness function should be modified in the course of evolution: in the early phase a fitness function should stress the learning ability and, in the second phase, when evolved NNs are able to learn the intended task, the fitness function should take into account a size of evolved NNs. In generality, we have to consider all features of a 'good solution' and, defining the fitness function, we should assure a balance between these fetures.

The third recommendation comes from comparison of different approaches (GAs, GAPP and GEP). Author's and cited GEP studies seem to discourage from usage of complex EAs as a tool for NNs design. The reason does not lie in the produced results. For example, GAPP produces good results for a number of tested problems, its efficiency is high. The complex algorithms usually have more parameters that have to be established, usually by users. It causes that they are not user-friendly, require knowledge, intuition and experience. Ferreira ([4]) concludes that GEP is promising technique, but seeing on the results it seems that the efficiency is too low: 30% – 70% for XOR problem and 4% – 6% for multiplexer (the most complex problem solved by GEP). Taking into account personal experience and the literature, the next recommendation can be formulated:

Advanced, complex models of EAs can be used to desing a NN that has to learn difficult task, but, in such situations, one has to put a great effort into understanding influence *of particular parameters on the efficiency of the used EA. For simpler problems, or if a user is less experienced, it is better to use a simpler evolutionary approach.*

Let us comeback to the second question. The commonly known rule says that knowing more, we better know how much we do not know. According to this rule, having in mind the development of fully automatic and efficient method of NNs design, we can formulate directions of further research.

The first subject of further research should be a *joining of advantages of two ways of NNs coding schemata: direct and indirect coding.* One possible approach is to start evolution of NNs using indirect (grammar) coding. When evolved NNs are able to learn an indended task, the evolution should be continued with NNs encoded using direct coding. It is difficult to predict an effect of such approach, becouse the quality space of NNs over the architectures is strongly deceptive: similar structures reveal different abilities, and inversely. It seems that only experimental studies can verify this idea.

The second subject of further study results from experiments with XOR problem. Backpropagation learning method was not able to learn the smalest NN, that was developed by a GA with evolutionary learning. As we know, a GA is able to escape from the local optimum. Majority of earlier experiments have suggested that evolutionary training usually required long time of evolution and had problems with tuning searched weights. But, from the other hand, backpropagation always goes to the nearest, often local optimum. Relatively good results have been observed when the result of evolutionary training have been tuned by backpropagation method. It seems that this problem requires further research. For large NNs a number of weights is very high, backpropagations find local optimum but a GA is inefficient searching multidimensional, multimodal weight space. *An approach with usage other, non-populational metaheuristics, e.g., Simulation Annealing or tabu search*, comes to mind.

Designing NNs we try to take into account a size of NN and its learning error. *The next idea is usage of multiobjective approach – search of Pareto optimal set of NNs seems to be worth further research.*

Next subject is connected with different architectures of NNs. To diagnose of specific problems connected with designing different architectures of NNs, i.e., spiking NNs, and working out an automatic method of such NNs design, seem to be a big challenge.

Although the research on automation of NNs design are conducted for many years, we still do not have satisfactory method. The problem requires further study. It seems that developing a good, fully automatic technique of Neural Network designing is still before us.

Bibliography

[1] L. Altenberg. Evolving better representation through selective genome growth. Proceedings of the IEEE 1994 World Congress on Computational Intelligence, pp. 182–187, 1994.

[2] S. Bornholdt, D. Graudenz. General Asymmetric Neural Networks and Structure Design by Genetic Algorithms. Neural Networks, Vol.5, pp. 327–334, 1992.

[3] C. Ferreira. Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence. http://www.gene-expression-programming.com/gep/Books/. 2006.

[4] C. Ferreira. Gene Expression Programming: A New Adaptive Algorithm for Solving Problems. Complex Systems. Vol. 13 (2), pp. 87–129, 2001.

[5] S.A. Harp, T. Samad , and A. Guha. Towards the Genetic Synthesis of Neural Networks. Proceeding of the third International Conference on GAs. George Mason University, United States, pp. 360–369, 1989.

[6] J.R. Koza. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press. 1992.

[7] H. Kwasnicka, M. Kilinska. Application of Genetic Algorithm to Neural Networks Design. Proceedings of the 12th Int. Conference on Systems Science, Vol. 1. System Theory. Control Theory. Wroclaw, pp. 164–168, 1995.

[8] H. Kwasnicka. Evolutionary Approach to Artificial Neural Network Design - Good Way or Blind Alley? Proceedings of the III International Mendel Conference on Genetic, Optimization Problems, Fuzzy Logic, Neural Networks, Rough Sets, Czech Republic, pp. 342–347, 1997.

[9] H. Kwasnicka. Evolutionary Computations in Artificial Intelligence (in Polish). Oficyna Wydawnicza PWr., Wroclaw 1999.

[10] H. Kwasnicka, S. Rynkiewicz. Neural Networks Design Using a GA with Pleiotropy Effect. Proceedings of the 10th International Conference on System Modelling Control. Poland, pp. 437–442, 2001.

[11] H. Kwasnicka. Hybrid Systems – Combining Neural Networks and Evolutionary Algorithms (in Polish). Proceedings of the Workshop on Evolutionary Algorithm. Gdynia Maritime University, R Smierzchalski (Ed.), Poland, pp. 13–15, 2003.

[12] H. Kwasnicka., M. Paradowski. Selection Pressure and an Efficiency of Neural Network Architecture Evolving. Proceedings of the 7 International Conf. on Artificial Intelligence and Soft Computing. LNCS No. 3070, Springer, pp. 444–449, 2004.

[13] H. Kwasnicka., M. Paradowski. Efficiency Aspects of Neural Network Architecture Evolution Using Direct and Indirect Encoding. Adaptive and natural computing algorithms. Eds B. Ribeiro [et all]. Portugal, Springer, pp. 405–408, 2005.

[14] L.R. Medsker. Hybrid Intelligent Systems. Kluver Academic Publishers. 1995.

[15] J.D. Montana, L. Davis. Training Feedforward Neural Networks Using Genetic Algorithms. BBN Systems and Technologies Corp. pp. 762–767, 1989.

[16] D. Murray. Tuning neural networks with genetic algorithms. AI Expert, 9(6), pp. 26–31, 1994.

[17] J.D. Schafer Chafer , Whitley D., Eshelman L.J. Combinations of Genetic Algorithms and Neural Networks: A Survey of the State International Workshop on Combinations of Genetic Algorithms and Neural Networks. Baltimore, MD, USA, pp. 1–37, 1992.

[18] D. Whitley. Genetic Algorithms and Neural Networks. Generic Algorithms in Engineering and Computer Science. G. Winter and J. Periaux and M. Galan and P. Cuesta (Eds.), John Wiley and Sons, Ltd., pp. 203–216, 1995.

[19] X. Yao. Evolving Artificial Neural Networks. Proceedings of the IEEE, 87(9), pp. 1423–1447, 1999.