computing@computingonline.net
www.computingonline.net

ISSN 1727-6209
International Scientific
Journal of Computing

FORMING EVOLUTIONARY DESIGN OF NEURAL NETWORKS WITH DIFFERENT NODES

Eva Volna

University of Ostrava,
30th Dubna st. 22,
701 03 Ostrava, Czech Republic
e-mail: eva.volna@osu.cz, http://www.osu.cz

Abstract: *Evolution in artificial neural networks (e.g. neuroevolution) searches through the space of behaviours for a network that performs well at a given task. Here is presented a neuroevolution system evolving populations of neurons that are combined to form the fully connected multilayer feedforward neural network with fixed architecture. In this article, the transfer function has been shown to be an important part of architecture of the artificial neural network and have significant impact on an artificial neural network's performance. In order to test the efficiency of described method, we applied it to the pattern recognition problem and to the alphabet coding problem.*

Keywords: *Neuroevolution, multilayer feedforward network, pattern recognition problem, alphabet coding problem.*

1. NEUROEVOLUTION

Neuroevolution represents a combination of neural networks and evolutionary algorithms (e.g. the genetic algorithm) where neural networks are the phenotype being evaluated. The genotype is a compact representation that can be translated into an artificial neural network. Evolution has been introduced into artificial neural networks at roughly three different levels: connection weights, architectures, and learning rules. The evolution of connection weights provides a global approach to connection weights training, especially when gradient information of the error function is difficult or costly to obtain. Due to the simplicity and generality of the evolution and the fact that gradient-based training algorithms often have to be run multiple times in order to avoid being trapped in a poor local optimum, the evolutionary approach is quite competitive. The evolution of architectures enables artificial neural networks to adapt their topologies to different tasks without human intervention and thus provides an approach to automatic artificial neural network design. Simultaneous evolution of artificial neural network architectures and connection weights generally produces better results. The evolution of learning rules in artificial neural networks can be used to allow an artificial neural network to adapt its learning rule to its environment. In a sense, the evolution provides artificial neural network with the ability of learning to learn.

Evolutionary algorithms are the term for different approaches as of using the models of evolutionary processes, which have nothing common with biology. They try to use the conception of driving forces of organism's evolution for optimization purposes. Evolutionary algorithms refer to a class of population-based stochastic search algorithms that are developed from ideas and principles of natural evolution. Fogel [1] gives a good introduction to various evolutionary algorithms for optimization. One important feature of all these algorithms is their population-based search strategy. Individuals in a population compete and exchange information with each other in order to perform certain tasks.

The choice of the right representation of individuals and their fitness create the essence of the advantageousness of the evolutionary algorithm, which depends on the selection of suitable choice of evolutionary algorithm and its appropriate operators. Individual within the evolutionary algorithm are then the problem solution. If a new solution is better, it substitutes the previous one. Optimization will be considered here as a synonym for minimization [2]. This is not a problem because of going in search the function maximum is equivalent to going in search of function minimum multiplied by -1.

Global search procedures such as evolutionary algorithms are usually computationally expensive. It would be better not to employ evolutionary

algorithms at all three levels of evolution. It is, however, beneficial to introduce global search at some levels of evolution, especially when there is little prior knowledge available at that level and the performance of the artificial neural network is required to be high, because the trial-and-error or heuristic methods are very ineffective in such circumstances. With the increasing power of parallel computers, the evolution of large artificial neural networks becomes feasible. Not only can such evolution discover possible new artificial neural network architectures and learning rules, but it also offers a way to model the creative process as a result of artificial neural network's adaptation to a dynamic environment.

2. EVOLUTION OF NODE TRANSFER FUNCTIONS

The architecture of artificial neural network includes its topological structure, i.e., connectivity, and the transfer function of each neuron in the artificial neural network. Architecture design is crucial in the successful application of artificial neural networks because the architecture has significant impact on a network's information processing capabilities. Up to now, architecture design is still very much a human expert's job. It depends heavily on the expert experience and a tedious trial-and-error process. There is no systematic way to design a near-optimal architecture for a given task automatically. Design of the optimal artificial neural network architecture can be formulated as a search problem in the architecture space where each point represents some architecture. Given some performance (optimality) criteria about architectures (e.g., lowest training error, lowest network complexity), the performance level of all architectures forms a discrete surface in the space. The optimal architecture design is equivalent to finding the highest point on this surface.

The discussion on the evolution of architectures so far only deals with the topological structure of architecture. The transfer function of each node in the architecture has been usually assumed that is fixed and predefined by human experts, at least for all the nodes in the same layer.

In principle, transfer functions of different neurons in artificial neural networks can be different (e.g. hard-limiting threshold function, a Gaussian function, sigmoid functions etc.) and decided automatically by an evolutionary process, instead of assigned by human experts. The decision on how to encode transfer functions in chromosome depends on how much prior knowledge and computation time is available. In general, neurons within a group, like a layer, in an artificial neural network tend to have the same type of transfer function with possible difference in some parameters, while different groups of neurons might have different types of transfer function. This suggests some kind of indirect encoding method, which lets developmental rules to specify function parameters if the function type can be obtained through evolution, so that more compact chromosomal encoding and faster evolution can be achieved. Little work has been only done on the evolution of node transfer function up to now. Mani proposed a modified backpropagation, which performs gradient descent search in the weight space as well as the transfer function space [3], but connectivity of artificial neural networks was fixed. Lovel and Tsoi investigated the performance of neocognitrons with various S-cell and C-cell transfer functions, but did not adopt any adaptive procedure to search for an optimal transfer function automatically [4]. Stork et al. [5] were the first to apply evolutionary algorithms to the evolution of both topological structures and node transfer functions even though only simple artificial neural networks with seven nodes were considered. The transfer function was specified in the structural genes in their genotypic representation. It was much more complex than the usual sigmoid function because authors in [5] tried to model biological neurons. White and Ligomenides [6] adopted a simpler approach to the evolution of both topological structures and node transfer functions. For each individual (i.e. the artificial neural network) in the initial population, 80% nodes in the artificial neural network used the sigmoid transfer function and 20% nodes used the Gaussian transfer function. The evolution was used to decide the optimal mixture between these two transfer functions automatically. The sigmoid and Gaussian transfer function themselves were not evolvable. No parameters of the two functions were evolved. Liu and Yao [7] used evolutionary programming to evolve artificial neural networks with both sigmoidal and Gaussian nodes. Rather than fixing the total number of nodes and evolve mixture of different nodes, their algorithm allowed growth and shrinking of the whole artificial neural network by adding or deleting a node (either sigmoidal or Gaussian). The type of node added or deleted was determined at random. Authors in [8, 9, 10] went one step further. They evolved topology of artificial neural network, node transfer function, as well as connection weights for projection neural networks. Sebald and Chellapilla [11] used the evolution of node transfer function as an example to show the importance of evolving representations. Representation and search are the two key issues in problem solving. Co-evolving solutions and their representations may be an effective way to tackle some difficult problems

where little human expertise is available. In principle, the difference in transfer functions could be as large as that in the function type, e.g. that between a hard limiting threshold function and Gaussian function, or as small as that in one of parameters of the same type of function, e.g. the slope parameter of the sigmoid function. One point worth mentioning here is the evolution of both connectivity and transfer functions at the same time [5] since they constitute a complete architecture. Encoding connectivity and transfer functions into the same chromosome makes it easier to explore nonlinear relations between them. Many techniques used in encoding and evolving connectivity could equally be used here.

The evolutionary approaches discussed so far in designing artificial neural network architecture evolve architectures only, without any connection weights. Connection weights have to be learned after a near-optimal architecture is found. This is especially true if one uses the indirect encoding scheme of network architecture. One major problem with the evolution of architectures without connection weights is *noisy fitness evaluation* [18]. In other words, fitness evaluation is very inaccurate and noisy because a phenotype's (i.e., an artificial neural network with a full set of weights) fitness was used to approximate its genotype's (i.e., an artificial neural network without any weight information) fitness. We want to optimize the genotype so that it can perform well regardless of initial connection weights, but we can only approximate such optimization by examining phenotypes with limited sets of initial connection weights of a virtually indefinite number of sets. There are two major sources of noise [7]:

- The *first* source is the random initialization of the weights. Different random initial weights may produce different training results. Hence, the same genotype may have quite different fitness due to different random initial weights used in training.
- The *second* source is the training algorithm. Different training algorithms may produce different training results even from the same set of initial weights. This is especially true for multimodal error functions. For example, backpropagation may reduce an artificial neural network's error to 0.05 through training, but an evolutionary algorithm could reduce the error to 0.001 due to its global search capability.

In order to reduce such noise, an architecture usually has to be trained many times from different random initial weights. The average result is then used to estimate the genotype's mean fitness. This method increases the computation time for fitness evaluation dramatically. It is one of the major reasons why only small artificial neural network were evolved in this way. In essence, the noise is caused by the one-to-many mapping from genotypes to phenotypes. It is clear that the evolution of architectures without any weight information has difficulties in evaluating fitness accurately. One way to alleviate this problem is to evolve artificial neural network architectures and connection weights simultaneously [16, 19]. In this case, each individual in a population is a fully specified artificial neural network with complete weight information. Since there is a one-to-one mapping between a genotype and its phenotype, fitness evaluation is accurate.

3. FIXED-TOPOLOGY NEUROEVOLUTION

Fixed-topology methods require a human to decide the right topology for a problem and most of them optimize connection weights only. Some highest performing neuroevolution systems (e.g. a system that performs better than any other systems on benchmark tasks) realize a more parametric evolution. The Symbiotic Adaptive Neuro-Evolution algorithm (SANE) [12] deals with a population of individual neurons, each of which is represented by a numerical vector defining the weight of its connections to each of the input and output neurons. In addition, there is a population of network blueprints consisting of pointers to neurons in the population. Individual networks are built out of neurons' subsets that are specified by one of blueprints, and the performance of the network as a whole is assigned to both the blueprint and to each individual neuron contributing to the network. The network blueprints are also evolved, with crossover recombination between network representations, mutations where half of the pointers are changed to offspring of the neurons to which they previously pointed, and mutations where a small fraction of the pointers are set to completely random neurons. Similarly, the top performing neurons themselves are also recombined and mutated to produce new neurons. The Enforced Sub-Population (ESP) [13], variant of SANE, is based on the special sort of separation. Rather than having one large pool of neurons with network blueprints, ESP maintains a separate population of neurons for each position in the network. Building a network then consists of selecting exactly one neuron from each of these subpopulations. Since the populations are kept separate during the creation of the next generation, each population is able to focus on a particular function more quickly, and networks are less likely to have redundant neurons. Recently, in [14] is successfully applied a special evolutionary strategy

called Evolution Strategy with Covariance Matrix Adaptation (CMA ES) to the evolution of fixed-topology neural networks. This method keeps track of correlations between changes of different weights in the network and fitness. Based on this information, the CMA-ES changes the covariance matrix of the weight mutation distribution so that it becomes more biased towards what were so far the most promising directions of search.

In the article, the transfer function has been shown to be an important part of architecture of the artificial neural network, one has significant impact on artificial neural network's performance. Here is presented a neuroevolution system evolving populations of neurons that are combined to form the fully connected multilayer feedforward neural networks with fixed architecture. Neuroevolution evolves transfer functions of each neuron in hidden and output layers of the network. The system maintains diversity in the population, because a dominant neural phenotype is likely to end up in the same network more than once. As several different types of neurons are usually necessary to solve a problem, networks with too many copies of the same neuron are likely to fail. The dominant phenotype then loses fitness and becomes less dominant. The system works well because it makes sure neurons get the credit they deserve, unlike some other neuroevolution techniques, where bad neurons can share in a good network or good neurons can be brought down by their network. It also works by decomposing the task, breaking the search into smaller, more manageable parts.

In the following is described a method of automatic search the node transfer function architecture in multilayer feedforward neural network: First, we must propose neural network architecture before the main calculation. We get the number of input (m) and output (o) neurons from the training set. Next, we have to define the number of hidden neurons (h) that is very confounding issue, because it is generally more difficult to optimize large networks than small ones. Thereafter the process of evolutionary algorithms is applied. Chromosomes are generated for every individual from the initial population as follows, see Fig. 1:

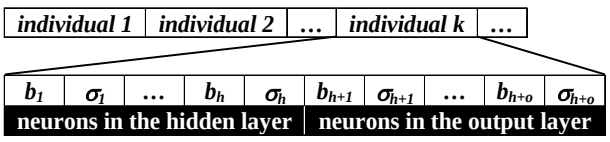


Fig. 1 – Population of individuals and their chromosomes.

Symbols b_i ($i = 1, \ldots, h+o$) refers to varies types of activation functions [15]:

- $b_i = 1$, if the activation function is a *binary sigmoid function*:

$$f(x) = \frac{1}{1 + \exp(-\sigma\, x)} \tag{1}$$

where σ is the steepness parameter, which value is set in the initial population randomly, e.g. $\sigma_i \in \{1,2,3,4,5,6,7\}$.

- $b_i = 2$, if the activation function is a binary *step function with threshold* θ:

$$f(x) = \begin{cases} 1 & if\ x \geq \theta \\ 0 & if\ x < \theta \end{cases} \tag{2}$$

the steepness parameter σ_i is not define here thus we assigned value 0 to it.

- $b_i = 3$, if the activation function is a *Gaussian function*:

$$f(x) = \exp(-x^2) \tag{3}$$

the steepness parameter σ_i is not define here thus we assigned value 0 to it.

- $b_i = 4$, if the activation function is a *saturated linear function*:

$$f(x) = \begin{cases} 0 & if\ x < 0 \\ x & if\ 0 \leq x \leq 1 \\ 1 & if\ x > 1 \end{cases} \tag{4}$$

the steepness parameter σ_i is not define here thus we assigned value 0 to it.

Next, we calculate an error value (E) between the desired and the real output after defined partial training with genetic algorithms. Adaptation of each individual starts with randomly generated weight values that are the same for each neural network in the given population. On the basis of it is calculated a fitness function for every individual as follows:

$$Fitness_i = E_{max} - E_i. \tag{5}$$

for $i = 1, \ldots, N$; where E_i is error for the *i-th* network after a partial adaptation, E_{max} is a maximal error for the given task, $E_{max} = o \times pattern$ (o is number of output neurons and *pattern* is number of patterns), N is the number of individuals in the population.

All of the calculated fitness function values of the two consecutive generations are sorted descending

and the neural network representation attached to the first half creates the new generation. For each fitness function is calculated the probability of reproduction its existing individual by standard method. One-point *crossover* was used to generate two offsprings. If the input condition of *mutation* is fulfilled (e.g. if a random number is generated, that is equal to the defined constant), one of the individuals is randomly chosen. There is randomly replaced one place in its genetic representation by a random value from the set of permitted values. The process of the evolutionary algorithm is ended, if the saturation parameter τ[1] is greater then define value, i.e. the population is composed only from similar types of individuals.

4. EXPERIMENTAL TASKS

In order to test the efficiency of described method, we applied it to the pattern recognition problem and to the alphabet coding problem that exists in cryptography.

Pattern recognition problem: For coding input examples the scheme from Fig. 2 is needed. If in an appropriate place the connection exists (e.g. on side with number 1, ..., 7), than the input chain representation is 1. If a connection does not exist, we have 0 in the appropriate place. Input vector contains thus seven bits. Output vector has got four bits and codes binary values of the input chain number representation.

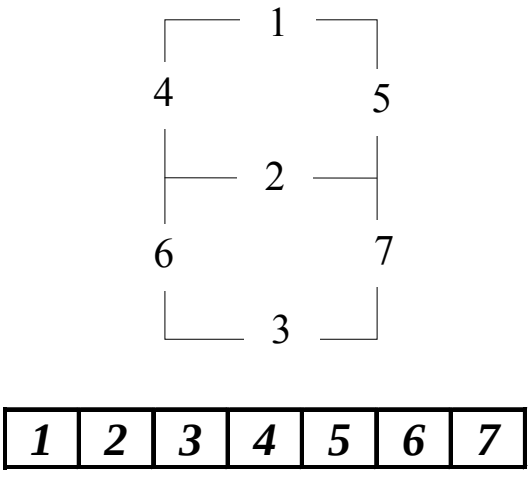


Fig. 2 – Scheme for coded examples to input chain of bits.

Neural network was trained to the following examples (see Fig. 3):

1011111→0000
0000101→0001
1110110→0010
1110101→0011
0101101→0100
1111001→0101
1111011→0110
1000101→0111
1111111→1000
1111101→1001

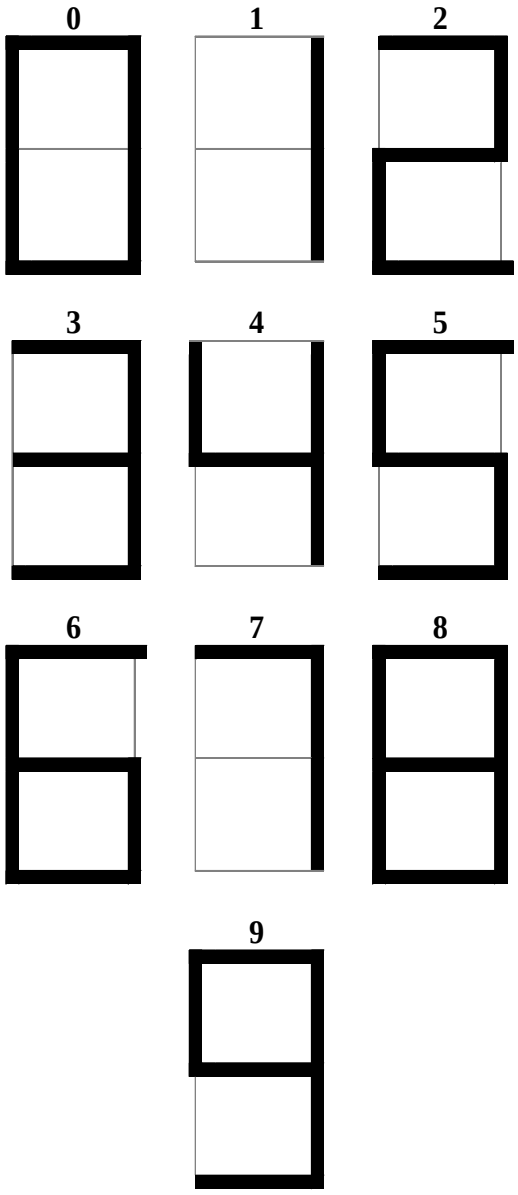


Fig 3 – The set of patterns (the training set).

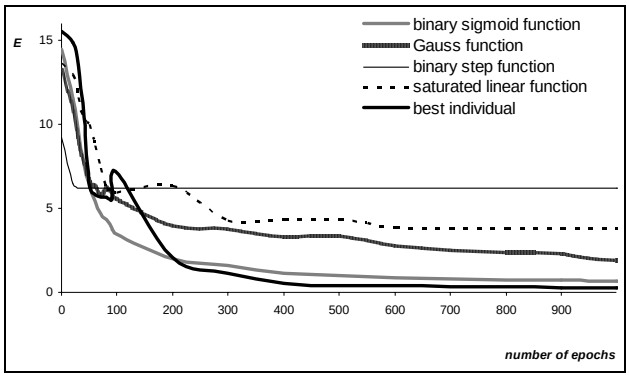
[1] τ is equal to a number of the same values at the same positions in chromosomes.

Table 1. The set of patterns (the training set).

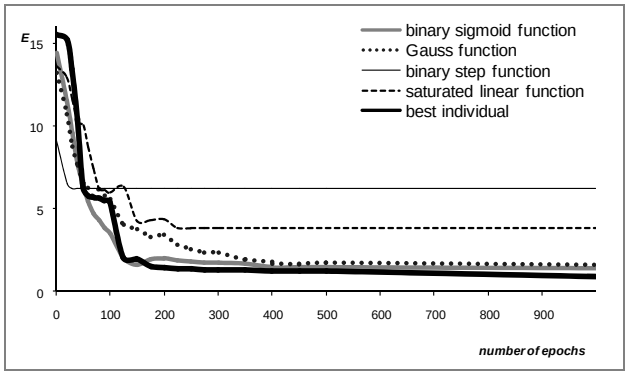
THE PLAIN TEXT			THE CIPHER TEXT
Char	**ASCII code (DEC)**	**The chain of bits**	**The chain of bits**
a	**97**	**00001**	**000010**
b	**982**	**00010**	**100110**
c	**99**	**00011**	**001011**
d	**100**	**00100**	**011010**
e	**101**	**00101**	**100000**
f	**102**	**00110**	**001110**
g	**103**	**00111**	**100101**
h	**104**	**01000**	**010010**
i	**105**	**01001**	**001000**
j	**106**	**01010**	**011110**
k	**107**	**01011**	**001001**
l	**108**	**01100**	**010110**
m	**109**	**01101**	**011000**
n	**110**	**01110**	**011100**
o	**111**	**01111**	**101000**
p	**112**	**10000**	**001010**
q	**113**	**10001**	**010011**
r	**114**	**10010**	**010111**
s	**115**	**10011**	**100111**
t	**116**	**10100**	**001111**
u	**117**	**10101**	**010100**
v	**118**	**10110**	**001100**
w	**119**	**10111**	**100100**
x	**120**	**11000**	**011011**
y	**121**	**11001**	**010001**
z	**122**	**11010**	**001101**

Alphabet coding problem: Neural networks can be also used in encryption or decryption algorithms, where parameters of adapted neural networks are included to cipher keys. Cipher keys must have several heavy attributes. The best one is the singularity of encryption and cryptanalysis [17]. Encryption is a process in which we transform the open text (e.g. news) to cipher text according to rules. Cryptanalysis of the news is the inverse process, in which the receiver of the cipher transforms it to the original text. The open text is composed from alphabet characters, digits and punctuation marks. The cipher text has usually the same composition as the open text. We worked [15] with multilayer neural networks, which topologies were based on the training set (see Table 1). The chain of chars of the plain text in a training set is equivalent to a binary value that is 96 less than its ASCII code. The cipher text is then a random chain of bits.

The initial population in both experiments contains 30 three-layer feedforward neural networks. Each network architecture is 7 - 7 - 4 for *pattern recognition problem*, and 5 - 5 - 6 for *alphabet coding problem* [15], because both problems are not linearly separable and therefore we cannot use neural network without hidden layer of neurons. All nets are fully connected. We use the genetic algorithm with the following parameters: probability of mutation is 0.01 and probability of crossover is 0.5. The saturation parameter τ is 95%. Adaptation of each neural network in given population starts with randomly generated weight values that are the same for each neural network in the population. We also used genetic algorithms with the same parameters for the partial neural network adaptation, where number of generations for a partial adaptation was 500. Their chromosome representation is described in [16].



(a)



(b)

Fig. 4 – The Error function history: (a) *pattern recognition problem,* (b) *alphabet coding problem*.

History of the error functions is shown in the Fig. 4. There are shown average values of error functions in the given population. Other numerical simulations gave very similar results. The "*binary*

sigmoid function" represents an average value after adaptation with the binary sigmoid activation function consecutively with all steepness parameters $\sigma = \{1,2,3,4,5,6,7\}$. The "*binary step function*" represents an adaptation with the binary step activation function (with the threshold θ), the "*saturated linear function*" represents an adaptation with the saturated linear activation function, and "*Gaussian activation*" represents an adaptation with the Gaussian activation function. Each of these mentioned representations is associated with all neurons in given neural network architecture. Opposite of this, the "*best individual*" represents an adaptation of the best individual in population, which chromosomes are the following, see Fig. 5:

b_1	σ_1	b_2	σ_2	b_3	σ_3	b_4	σ_4	b_5	σ_5	b_6	σ_6	b_7	σ_7
1	7	1	5	1	1	1	5	3	0	3	0	1	3

neurons in the hidden layer

b_8	σ_8	b_9	σ_9	b_{10}	σ_{10}	b_{11}	σ_{11}
3	0	1	7	1	2	1	6

neurons in the output layer

(a)

b_1	σ_1	b_2	σ_2	b_3	σ_3	b_4	σ_4	b_5	σ_5
1	5	1	7	1	1	3	0	1	5

neurons in the hidden layer

b_6	σ_6	b_7	σ_7	b_8	σ_8	b_9	σ_9	b_{10}	σ_{10}	b_{11}	σ_{11}
3	0	1	7	3	0	1	5	1	6	1	2

neurons in the output layer

(b)

Fig. 5 – The "best individual" chromosome in the last population: a) *pattern recognition problem,* (b) *alphabet coding problem.*

5. CONCLUSIONS

All networks solve the pattern recognition task resp. alphabet coding problem in our experiments, but artificial neural network with evolving transfer functions of each neuron works well, because several different types of neurons are usually necessary to solve a problem. We can see that the proposed technique is really efficient for the presented purpose, see the Fig. 3 or Table 1. Networks with too many copies of the same neuron work usually worse, see the Fig. 4.

Here, the transfer function is shown to be an important part of architecture of the artificial neural network and have significant impact on artificial neural network's performance. Transfer functions of different neurons can be different and decided automatically by an evolutionary process, instead of assigned by human experts.

In general, nodes within a group, like layer, in an artificial neural network tend to have the same type of transfer function with possible difference in some parameters, while different groups of nodes might have different types of transfer function.

6. REFERENCES

[1] D. B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* IEEE Press. New York, 1995.

[2] P. Hammerstein, E. H. Hagen, A.V. Herz, M.H. Herzel. Robustness: A key to evolutionary design. *Biol. Theory* 1(1) 90–93, 2006.

[3] G. Mani. Learning by gradient descent in function space. *Proc. of the IEEE Int. Conf. "System, Man, and Cybernetics"*, Los Angeles, CA, 1990, pp. 242–247.

[4] D. R. Lovell. A. C. Tsoi. *The Performance of the Neocognitron with Various S-Cell and C-Cell Transfer Functions*, Intell. Machines Lab., Dep. Elect. Eng., Univ. Queensland, Tech. Rep., Apr. 1992.

[5] D. G. Stork. S. Walker. M. Burns. B. Jackson. Preadaptation in neural circuits. *Proc. Int. Joint Conf. "Neural Networks"*, vol. I, Washington, DC, 1990, pp. 202–205.

[6] D. White. P. Ligomenides. GANNet: A genetic algorithm for optimizing topology and weights in neural network design. *Proc. Int. Workshop "Artificial Neural Networks (IWANN'93)"*, Lecture Notes in Computer Science, vol. 686. Berlin, Germany: Springer-Verlag, 1993, pp. 322–327.

[7] Y.Liu. X. Yao. Evolutionary design of artificial neural networks with different nodes. *Proc. 1996 IEEE Int. Conf. "Evolutionary Computation (ICEC'96)"*, Nagoya, Japan, pp. 670–675.

[8] A. Abraham. Meta learning evolutionary artificial neural networks. *Neurocomputing*, vol. (56) 1-38, 2004.

[9] F. H. F. Leung, H. K. Lam, S. H. Ling, P. K. S. Tam. Tuning of the structure and parameters of a neural network using an improved genetic algorithm. *IEEE Transactions on Neural Neworks*. Vol. 14 (1) 79- 88, 2003.

[10] P. Palmes, S. Usui. Robustness, Evolvability and Optimality in Evolutionary Neural Networks". *Biosystems*, vol. 82 (2) 168-188, 2005.

[11] A. V. Sebald, K. Chellapilla. On making problems evolutionarily friendly, part I: Evolving the most convenient representations. In V. W. Porto. N.Saravanan, D. Waagen. A. E. Eiben. (Eds.) *Evolutionary Programming VII: Proc. 7th Annu Conf. "Evolutionary Programming"*, vol. 1447 of Lecture Notes in Computer Science, Berlin, Germany: Springer-Verlag, 1998, pp. 271–280.

[12] R. Miikkulainen. Evolving neural networks. In *Proceedings of the 2007 GECCO Conference Companion on Genetic and Evolutionary Computation* GECCO '07. ACM, New York, NY, 2007, pp. 3415-3434.

[13] F. J. Gomez. R. Miikkulainen. Active guidance for a finless rocket through neuroevolution. *Proceedings of the Conference "Genetic and Evolutionary Computation (GECCO-2003)"*. Berlin: Springer Verlag. 2003.

[14] C. Igel. Neuroevolution for reinforcement learning using evolution strategies. In R. Sarker R.Reynolds. H. Abbass. K. C. Tan. B. McKay. D. Essam. T.Gedeon (eds.) "*Congress on Evolutionary Computation 2003 (CEC 2003)"* Piscataway, NJ: IEEE Press. 2003. pp. 2588–2595.

[15] E. Volna. Forming neural network design through evolution".. In K. Madani (ed.). *Proceedings of the 3th International Workshop on "Artificial Neural Networks and Intelligent Information Processing (*ANNIIP 2007)". In conjunction with ICINCO 2007. Angers, France 2007, pp. 13-20.

[16] Volná, E. "Learning algorithm which learns both architectures and weights of feedforward neural networks". *Neural Network World. Int. Journal on Neural & Mass-Parallel Comp. and Inf. Systems*. 8 (6): 653-664, 1998.

[17] S. Garfinger. *PGP: Pretty Good Privacy*. Computer Press, Praha 1998.

[18] E. Cantu-Paz. Adaptive sampling for noisy problems. In *Genetic and Evolutionary Computation Conference*, pages 947--958, Springer 2004.

[19] P. A., Castillo, J.J. Merelo, M. G. Arenas, and G. Romero. Comparing evolutionary hybrid systems for design and optimization of multilayer perceptron structure along training parameters. In *Information Sciences*, Vol 177 (14) 2884-2905, 2007.

Eva Volna *graduated at the Slovak Technical University in Bratislava and defended PhD. thesis with title "Modular Neural Networks".*

She has been working as an assistant professor at the Department of Computer Science, University of Ostrava (Czech Republic) from 1992.

Her interests include artificial intelligence, artificial neural networks, evolutionary algorithms, and cognitive science.

She is author of 37 publications - 9 articles in reviewed journals, 28 articles in proceedings of international and national conferences, 8 teaching texts and 3 graduation theses.