České vysoké učení technické v Praze
Fakulta elektrotechnická

Czech Technical University in Prague
Faculty of Electrical Engineering

RNDr. PaedDr. Eva Volná, PhD.

Neuroevoluce
Neuroevolution

Summary

Research on potential interactions between connectionist learning system, i.e. artificial neural networks, and evolutionary search procedures has attracted a lot of attention recently. Evolutionary artificial neural networks can be considered as a combination of artificial neural networks and evolutionary search procedures. We can distinguish among three kinds of evolution in evolutionary artificial neural networks, i.e. the evolution of connection weights, of architectures, and of learning rules. Then it reviews each kind of evolution in detail and analyses critical issues related to different evolutions. The review shows that although a lot of work has been done in the evolution of connection weights and of architecture, few attempts have been made to understand the evolution of learning rules. Interactions among different evolutions are seldom mentioned in the current research. However, the evolution of learning rules and its interactions with other kinds of evolution play a vital role in evolutionary artificial neural networks.

Souhrn

Studium interakcí mezi spojitými adaptivními systémy, tj. umělými neuronovými sítěmi a evolučními prohledávacími procedurami je téma stále aktuální. Evoluční umělé neuronové sítě představují kombinaci umělých neuronových sítí a evolučních prohledávacích procedur. Rozlišujeme zde tři typy evoluce, tj. evoluci váhových hodnot na spojeních mezi neurony, evoluci architektury a evoluci adaptačních pravidel. Postupně uvedeme detailní přehled každé z uvažovaných evolucí. K problematice zabývající se evolucí váhových hodnot na spojeních mezi neurony a k problematice evoluce architektury neuronové sítě lze nalézt velké množství publikací, avšak na téma evoluce parametrů adaptačních pravidel bylo napsáno pouze několik prací. V aktuálním výzkumu evolučních umělých neuronových sítí jsou rovněž jen řídce zmíněny interakce mezi různými typy evolucí, i když evoluce parametrů adaptačních pravidel a její vzájemná interakce s ostatními typy evolucí zde hraje významnou roli.

Klíčová slova: umělé neuronové sítě, evoluční algoritmy, adaptace, optimalizace.

Keywords: artificial neural networks, evolutionary algorithms, adaptation, optimization.

Contents

1 Introduction

Under **neuroevolution** we can understand the connection of evolutionary algorithms and artificial neural networks - that is the using of evolutionary algorithm properties in suggestion of artificial neural network architecture and upon work with them. This thesis concentrates on finding the suitable way of using evolutionary algorithms for optimizing the artificial neural network parameters.

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. They include [17]: evolution strategies, evolutionary programming, genetic algorithms etc. All these models work with random changes of submitted solutions, and they do not use optimized function derivatives; only the values of said function. Optimization will be considered here as a synonym for minimization. This is not a problem because going in search the function maximum is equivalent to going in search of function minimum multiplied by -1. 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 individual within the evolutionary algorithm is then the problem solution. If a new solution is better, it substitutes the previous one. 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. A general framework of evolutionary algorithms can be described by figure 1. *To achieve a better clearness all the algorithms will be described in „Pseudopascal".*

1. Generate the initial population *G(0)* at random, and set *i = 0*;
2. REPEAT
(a) Evaluate each individual in the population;
(b) Select parents from *G(i)* based on the fitness in *G(i);*
(c) Apply search operators to parents and produce offspring which form *G(i+1);*.
(d) *i = i + 1;*
3. UNTIL 'termination criterion' is satisfied.

Figure 1: A general framework of evolutionary algorithms.

Artificial neural networks are discussed in numerous publications, e.g. [2, 10]. An artificial neural network is characterized by its pattern of connections between the neurons (called its *architecture*), its method of determining the weights on the connections (called its *training*, or *learning algorithm*), and its *activation function*. Neural network architecture can be described as a directed graph in which each neuron i performs a transfer function f_i of the form (1):

$$y_i = f_i\left(\sum_{j=1}^{n} w_{ij} x_j - \theta_i\right), \tag{1}$$

where y_i is the output of the neuron i, x_j is the j^{th} input to neuron i and w_{ij} is the connection weight between neurons i a j. θ_i is the threshold (or bias) of the neuron i. Usually, the activation function f_i is nonlinear, such as a sigmoid, or Gaussian function. Artificial neural networks can be divided into feedforward and recurrent classes according to their connectivity. An artificial neural network is feedforward if there exists a method, which numbers all the neurons in the network such that there is no connection between a neuron with a large number and a neuron with a smaller number. All the connections are from neurons with small numbers to neurons with larger numbers. An artificial neural network is recurrent if such a numbering method does not exist. Learning in artificial neural networks is typically accomplished using examples. This is also called training in artificial neural networks. Learning in artificial neural networks can roughly be divided into supervised, unsupervised, and reinforcement learning. Supervised learning is based on direct comparison between the actual output of an artificial neural network and the desired correct output, also known as the target output. It is often formulated as the minimization of an error function such as the total mean square error between the actual output and the desired output summed over all available data (2).

$$E = \frac{1}{2}\sum_{j=1}^{m}\sum_{i=1}^{n}\left(y_i - t_i\right)_j^2, \tag{2}$$

where y_i is the actual output of the neuron i, t_i is the desired correct output of the neuron i, n is a number of output neurons, and m is a number of training patters. A gradient descent-based optimization algorithm such as backpropagation can be used to adjust connection weights in the artificial neural network iteratively in order to minimize the error (2). Reinforcement learning is a special case of supervised learning where the exact desired output is unknown. It is based only on the information of whether or not the actual output is correct. Unsupervised learning is solely based on the

correlations among input data. No information of correct output is available for learning. The essence of a learning algorithm is *the learning rule*, i.e., a weight-updating rule that determines how connection weights are changed. Examples of popular learning rules include the delta rule, the Hebbian rule, the competitive learning rule, etc. are discussed in numerous publications, e.g. [2, 10].

2 Evolution in artificial neural networks

Evolution has been introduced into artificial neural networks at roughly three different levels: connection weights, architectures, and learning rules. The *evolution of connection weights* introduces an adaptive and global approach to problem solution. 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. The *evolution of learning rules* can be regarded as a process of "learning to learn" in artificial neural networks, where the adaptation of learning rules is achieved through evolution.

2.1 The evolution of connection weights

The evolutionary approach to weight training in artificial neural networks consists of two major phases. The first phase means to decide the representation of connection weights. The second one means the evolutionary process simulated by evolutionary algorithms. A typical cycle of the evolution of connection weights is shown in figure 2.

1. **Decode each individual (a neural network) in the current generation into a set of connection weights and construct a corresponding artificial neural network with weights.**
2. **Evaluate each individual (a neural network) by computing its total mean square error between actual and target outputs (2), but other error functions can also be used. The fitness of an individual is determined by the error (e.g. the higher of error, the lower of fitness).**
3. **Select parents for reproduction based on their fitness** [12].
4. **Apply search operators, such as crossover / or mutation, to parents to generate offspring, which form the next generation.**

Figure 2: A typical cycle of the evolution of connection weights.

The most convenient representation of connection weights is, from evolutionary algorithm's perspective, **binary** string. In such a representation scheme, each connection weight is represented by a number of bits of a certain length. An artificial neural network is encoded by concatenation of all the connection weights of the network into the chromosome. The order of the concatenation is, however, essentially ignored, although it can affect the performance of evolutionary training, e.g. training time and accuracy [3]. The advantages of the binary representation lie in its simplicity and generality. It is straightforward to apply classical crossover (such as one-point or uniform crossover) and mutation to binary strings. The binary representation also facilitates digital hardware implementation of artificial neural networks since weights have to be represented in terms of bits in hardware with limited precision. A limitation of binary representation is the representation precision of discrete connection weights. It is still an open question how to optimize the number of bits for each connection weight, the range encoded, and the encoding method used although dynamic techniques could be adopted to alleviate the problem.

To overcome some shortcomings of the binary representation scheme, **real** numbers themselves proposed to represent connection weights directly, i.e. one real number per connection weight. It is discussed in numerous publications, e.g. [24, 28]. The chromosome is represented by the concatenation of these real numbers, where their order is important. As connection weights are represented by real numbers, each individual in an evolving population is a real vector. Standard search operators dealing with binary strings cannot be applied directly in the real representation scheme. In such circumstances, an important task is to design carefully a set of genetic operators, which are suitable for the real representation as well as artificial neural network's training, in order to improve the speed and accuracy of the evolutionary training. Single real numbers are often changed by average crossover, random mutation or other domain specific genetic operators. It is discussed in numerous publications, e.g. [9, 24]. The major aim is to retain useful functional blocks during evolution, i.e., to form and keep useful feature detectors in an artificial neural network.

Evolutionary algorithms can handle the global search problem better in a vast, complex, multimodal and no differentiable (connection weight) space. They are usually based on a global search algorithm, thus can escape from a local minimum, while a gradient descent algorithm can only find a local optimum in a neighborhood of the initial solution. An evolutionary algorithm sets no restriction on types of artificial neural networks being trained as long as a suitable fitness function can be defined properly, thus can deal with a wide range of artificial neural networks: recurrent artificial neural networks, high-order artificial neural networks, fuzzy artificial neural

networks etc. An assignment of the most acceptable evolutionary algorithm to a task represents always a big problem, because each search procedure is suitable only for a class of error (fitness) functions with certain types of landscape, the issue of what kind of search procedure is more suitable for which class of error (fitness) function is an important research topic of general interest. The efficiency of evolutionary training can be improved significantly by incorporating a local search procedure into the evolution, i.e., combining evolutionary algorithm's global search ability with local search's ability to fine tune. Evolutionary algorithms can be used to locate a good region in the space and a local search procedure is used to find a near-optimal solution in this region. *Hybrid training* has been used successfully in many application areas. The obtained results showed that the hybrid GA/BP approach was more efficient than if either the genetic or backpropagation algorithm alone [33] were used, because genetic algorithms are much better at local good initial weights than the random start backpropagation method. Similar work on the evolution of initial weights has also been done on competitive learning neural networks and Kohonen networks [36]. One of the problems faced by evolutionary training of artificial neural networks is the *permutation problem* [3, 32], also known as the *competing convention problem*. It is caused by the many-to-one mapping from the representation (genotype) to the actual artificial neural network (phenotype) since two artificial neural networks that order their hidden neurons differently in their chromosomes will still be equivalent functionally. In general, any permutation of the hidden neurons will produce functionally equivalent artificial neural networks with different chromosome representations. The permutation problem makes the crossover operator very inefficient and ineffective in producing good offspring. It is generally very difficult to apply crossover operators in evolving connection weights since they tend to destroy feature detectors found during the evolutionary process, because hidden nodes are in essence feature extractors and detectors. Separating inputs into the same hidden node far apart in the representation would increase the difficulty of constructing useful feature detectors because they might be destroyed by crossover operators.

2.2 The evolution of architectures

The architecture of an 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. Constructive / destructive algorithms are one of the many efforts made towards the automatic design of artificial neural network architecture. A *constructive algorithm* starts with a minimal network (e.g. network with minimal number of hidden layers, neurons, and connections) and adds new layers, neurons, and connections when necessary during training while a *destructive algorithm* does the opposite, i.e., starts with the maximal network and deletes unnecessary layers, neurons, and connections during training. These methods are susceptible to becoming trapped at local optima, and in addition, they only investigate restricted topological subsets rather than the complete class of network architectures. The 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. 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. There are several characteristics of such a surface, which make the evolutionary algorithms a better candidate for searching the surface than the constructive and destructive algorithms. These characteristics are [23] the following:

- the surface is *infinitely large* since the number of possible neurons and connections is unbounded;
- the surface is *no differentiable* since changes in the number of neurons or connections are discrete and can have a discontinuous effect on artificial neural network's performance;
- the surface is *complex* and *noisy* since the mapping from an architecture to its performance is indirect and dependent on the evaluation method used;
- the surface is *deceptive* since similar architectures may have quite different performance;
- the surface is *multimodal* since different architectures may have similar performance.

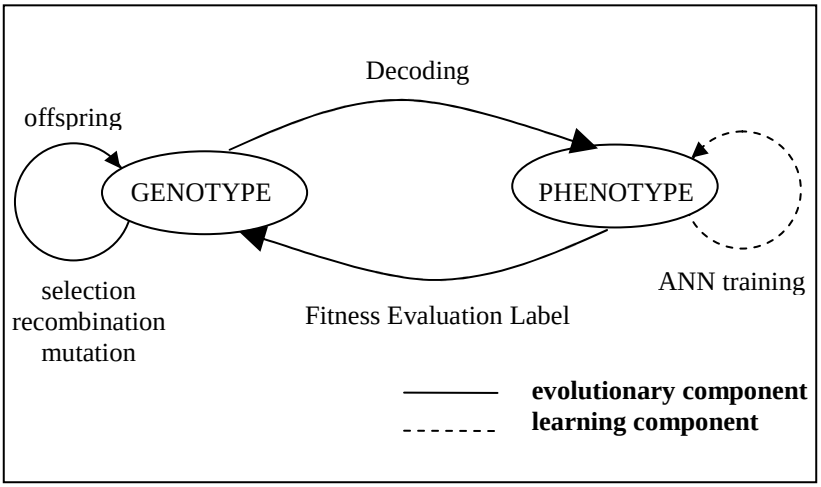

Figure 3: Process of evolutionary design of artificial neural networks.

Similar to the evolution of connection weights, two major phases involved in the evolution of architectures are the genotype representation scheme of architectures and the evolutionary algorithm used to evolve artificial neural network architectures. But the problem now is not whether to use a binary representation or a real one, since we only deal with discrete values, a binary representation is required. A key issue here is to decide how much information about architecture should be encoded into a representation. As we apply discrete values, we use usually a binary representation, i.e. matrices or graphs. This kind of representation is called *direct encoding scheme*. It can be seen that only parameters of a connectivity pattern, instead of each individual connection, is specified by a chromosome in bigger networks [14]. It is possible to choose more complicated way of encoding to optimize huge network architectures, where only the most important parameters or features of architecture are represented, such as the number of neurons, the number of connections, and the type of transfer functions. Other details of the architecture are left to learning (training) process to decide. Such kind of representation is called the *indirect encoding scheme.* The net architecture is then specified by developmental rules, by sentences of a formal language, etc. The evolution of neural network architecture, regardless of the chosen type of its representation (kind of encoding) is shown in figure 3 and figure 4. The stochastic optimization algorithms are in principle the only systematic approach to optimization of neural network architecture. Continuous research on evolving neural network architecture has been carried out in the recent years, e.g. [4, 20, 35]. Most of the research outputs have been

concentrated on the evolution of artificial neural network topological structures. Several of them have been focused to the evolution of node transfer functions, e.g. [34, 39].

1. **Decode each individual in the current generation into architecture with necessary details.**
2. **Train each artificial neural network with the decoded architecture by the predefined learning rule starting from different set of random initial connection weights, if any, learning rule parameters.**
3. **Compute the fitness of each individual (encoded architecture) according to the training results and other performance criteria such as the complexity of the architecture.**
4. **Select parents from the population based on their fitness.**
5. **Apply search operators to the parents and generate offspring which form the next generation.**

Figure 4: A typical cycle of the evolution of architectures.

Two different approaches have been taken in the **direct encoding scheme**. The first separates the evolution of architectures from that of connection weights. The second approach evolves architectures and connection weights simultaneously. Both approaches require determining the fixed network topology before calculation. Next, we will focus on the first approach. In the direct encoding scheme, each connection between neurons is directly specified by its binary representation. For example, an $N \times N$ matrix $C = (c_{ij})_{N \times N}$ can represent an artificial neural network architecture with N neurons, where c_{ij} indicates presence or absence of the connection from neuron i to neuron j. We can use $c_{ij} = 1$ to indicate a connection and $c_{ij} = 0$ to indicate no connection. Each matrix C has a direct one-to-one mapping to the corresponding artificial neural network architecture. The binary string representing an architecture is the concatenation of rows (or columns) of the matrix. The direct encoding scheme is quite straightforward to implement. It is very suitable for the precise and deterministic search of compact artificial neural network architecture, since a single connection can be added or removed from the artificial neural network easily. One potential problem of the direct encoding scheme is *scalability*. A large artificial neural network would require a very large matrix and thus increase in the computation time of evolution. In [16] is shown that the length of the genotype is proportional to the complexity of the corresponding phenotype, and the space to be searched by the evolutionary process increases exponentially with the size of the network. One way to cut down the size of matrices is to use domain

knowledge to reduce the search space. For example, if a complete interconnection is to be used in a feedforward artificial neural network between two neighboring layers, its architecture can be encoded by just the number of hidden layers and nodes in each hidden layer (e.g. *Layer-based encoding*), therefore the length of chromosome can be reduced considerably. However, doing so requires sufficient domain knowledge and expertise, which are difficult to obtain in practice. We also run risk of loss of certain very good solutions when restricting the search space manually. Another problem of direct encoding schemes is the impossibility to encode repeated structures (such as network composed of several sub-networks with similar local connectivity) in a compact way. In one-to-one mappings, in fact, elements that are repeated at the level of the phenotype must be repeated at the level of the genotype as well. This does not only affect the length of the genotype and the corresponding search space. A full genetic specification of a phenotype with repeated structures, in fact, implies that adaptive changes affecting repeated structures should be independently rediscovered through changes introduced by the genetic operators. The next problem to be considered in binary representation is that of *variable-length genomes* [31]. For example, where two parents have different topologies, it is not obvious how their offspring should be formed. When determining which nodes and connections the offspring should inherit, it would be helpful to know which subnetworks from the parents perform the same functions, and which represent disjoint concepts. Unfortunately, this information is not readily apparent from the two different topologies. Handling a binary representation we strive to take advantages from combining different solutions, while we draw attention to the confrontation between the flexibility of representations and compatibility of genotypes. The said might be implemented e.g. thru using the mutation operators only and / or by specially defined crossing operator, which minimize the genome destruction after its application, and which fails to combine these parts of genome, which preserve functionality of different structures.

In order to reduce the length of the genotype representation of architectures, we can use the **indirect encoding scheme**, where only the most important characteristics of architecture are encoded in the chromosome. The details about each connection in an artificial neural network is either predefined according to prior knowledge or specified by a set of deterministic developmental rules. Many of the indirect neural networks encoding strategies are inspired by the *Lindenmayer systems* [18]. Some references to the related papers are in [8]. The typical approach of such encoding is *a grammatical encoding* [16], where evolutionary algorithms do not develop network architecture directly, but rules of formal grammatics, being subsequently used for generating the network topology.

The shift from the direct optimization of architectures to the optimization of developmental rules has brought some benefits, such as more compact genotype representation. The rules do not grow with the size of artificial neural networks, since the rule size does not change. The rule is usually described by a recursive equation or a generation rule is similar to a production rule in a production system with a *left-hand side* and a *right-hand side*. In [13], a genetic encoding scheme for neural networks based on a cellular duplication and differentiation process was proposed. Genomes are programs written in a specialized graph transformation language called the *grammar tree*, which is very compact. The genotype-to-phenotype mapping starts with a single cell that undergoes a number of duplication and transformation processes ending up in a complete neural network. In this scheme the genotype is a collection of rules governing the process of cell divisions (a single cell is replaced by two "daughter" cells) and transformations (new connections can be added and the strengths of the connections departing from a cell can be modified). In this model, therefore, connection links are established during the cellular duplication process. The author [13] also considered the case of genotypes formed by many trees where the terminal nodes of a tree may point to other trees. This mechanism allows the genotype-to-phenotype process to produce repeated phenotype structures (e.g. repeated neural sub-networks) by reusing the same genetic information, which saves space in genome and it is useful even for keeping the substructures when applying the crossover operator. We cannot be sure whether the subgraphs combined mutually are the right building blocks to create a functional offspring. The literature [11] introduces a new algorithm based on *Gene Expression Programming* that performs a total network induction using linear chromosomes of fixed length that map into complex neural networks of different sizes and shapes. The total induction of neural networks using gene expression programming requires further modification of the structural organization developed to manipulate numerical constants and domain-specific operators. The author explains how the chromosome may be modified, so that a complete neural network including the architecture, the weights and threshold values, could be totally encoded in a linear chromosome. The indirect encoding scheme is biologically more plausible as well as more practical, from the view point of engineering, than the direct encoding scheme although some fine-tuning algorithms might be necessary to further improve the result of evolution. Other techniques of indirect neural network encoding topology are listed in numerous publications, e.g. [22, 26, 30].

The representation of artificial neural network architectures always plays an important role in the evolutionary design of architectures. There is not a single method, which outperforms others in all aspects. The best

choice depends heavily on applications at hand and available prior knowledge. A problem closely related to the representation issue is the design of genetic operators. However, the use of crossover appears to be inconsistent, because crossover works the best when *building blocks* exist but it is unclear what a building block might be in an artificial neural network since the artificial neural networks are featured with a distributed (knowledge) representation. The knowledge in an artificial neural network is distributed among all the weights in the artificial neural network. Recombining one part of an artificial neural network with another part of another artificial neural network is likely to destroy both artificial neural networks. However, if artificial neural networks do not use a distributed representation but rather a localized one, such as radial basis function networks or nearest-neighbor multilayer perceptrons, crossover might be a very useful operator, see e.g. [1, 40]. In general, artificial neural networks using distributed representation are more compact and have a better generalization capability for most practical problems. As for the evolution of connection weights, thus even here we have to resolve the *permutation problem* (*competing convention problem*) that causes enormous redundancy in the architecture space. Unfortunately, no satisfying technique has been implemented to tackle this problem. An advantage of the evolutionary approach is that the fitness function can be defined easily in such a way that an artificial neural network with some special features is evolved. For example, artificial neural networks with a better generalization can be obtained if testing results, instead of training results, are used in their fitness calculations. A *penalty term* in the fitness function for complex connectivity can also help improve artificial neural network's generalization ability, besides the cost benefit, by reducing the number of neurons and connections in artificial neural networks.

The discussion on the evolution of architectures so far only deals with the topological structure of architecture. *The transfer function* of each neuron in the architecture has been assumed to be fixed and predefined by human experts yet. The transfer function has been shown to be an important part of artificial neural network architecture and have significant impact on artificial neural network's performance. In principle, transfer functions of different neurons in an artificial neural network 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 functions. 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.

2.3 Simultaneous evolution of architectures and connection weights

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*, see e.g. [6, 38]. 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 [38]:

1. 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.
2. 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, 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 networks 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, see e.g. [5, 33]. 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.

2.4 The evolution of learning rules

An artificial neural network training algorithm may have different performance when applied to different architectures. The design of training algorithms, more fundamentally the learning rules used to adjust connection weights, depends on the type of architectures and learning tasks under investigation. After selecting a training algorithm, there are still algorithm parameters, like the learning rate and momentum in backpropagation algorithms, which have to be specified. For example genetic algorithms are suitable for training artificial neural networks with feedback connections and deep feedforward artificial neural networks (with many hidden layers), while backpropagation is good at training shallow ones. At present, this kind of search for an optimal (near optimal) learning rule can only be done by some experts through their experience and trial-and-error. In fact, what is needed from an artificial neural network is its ability to adjust its learning rule adaptively according to its architecture and the task to be performed. In other words, an artificial neural network should learn its learning rule dynamically rather than have it designed and fixed manually. Since evolution is one of the most fundamental forms of adaptation, then said evolution may contribute to the development of appropriate type of the learning rule for given application; for which also the fact may be utilized that the relationship between evolution and learning is extremely complex. Various models have been proposed, but most of them deal with the issue of how learning can guide evolution and the relationship between the evolution of architectures and that of connection weights. Research into the evolution of learning rules is still in its early stages, see e.g. [19, 29]. This research is important not only in providing an automatic way of optimizing learning rules and in modeling the relationship between learning and evolution, but also in modeling the creative process since newly evolved learning rules can deal with a complex and dynamic environment. A typical cycle of the evolution of learning rules can be described by figure 5. The iteration stops when the population converges or a predefined maximum number of iterations has been reached.

The **adaptive adjustment of algorithmic parameters** through evolution could be considered as the first attempt of the evolution of learning rules, e.g. in [15] encoded backpropagation's parameters in chromosomes together with the artificial neural network architecture. There is a number of algorithms with an adaptive learning rate and momentum [21], where also a nonevolutionary approach is used. Further comparison between the two approaches would be quite useful. The evolution of algorithmic parameters is certainly interesting but it hardly touches the fundamental part of a training algorithm, i.e., its learning rule or weight-updating rule.

1. **Decode each individual in the current generation into a learning rule.**
2. **Construct a set of artificial neural networks with randomly generated architectures and initial connection weights, and train them using the decoded learning rule.**
3. **Calculate the fitness of each individual (encoded learning rule) according to the average training result.**
4. **Select parents from the current generation according to their fitness.**
5. **Apply search operators to parents to generate offspring, which form the new generation.**

Figure 5: A typical cycle of the evolution of learning rules.

Adapting a **learning rule** through evolution is expected to enhance the artificial neural network's adaptivity greatly in a dynamic environment. It is much more difficult to encode dynamic behaviours, like the learning rule, than to encode properties, like the architecture and connection weights, of an artificial neural network. The key issue here is how to encode the dynamic behavior of a learning rule into static chromosomes. Trying to develop a universal representation scheme, which can specify any kind of dynamic behaviors, is clearly impractical, let alone the prohibitive long computation time required searching such a learning rule space. Constraints have to be set on the type of dynamic behaviors, i.e., the basic form of learning rules being evolved in order to reduce the representation complexity and the search space. Two basic assumptions which have often been made on learning rules are [37]: a) weight updating depends only on local information such as the activation of the input neuron, the activation of the output neuron, the current connection weight, etc., and b) the learning rule is the same for all connections in an artificial neural network. A learning rule is assumed to be a linear function of these local variables and their products. That is, a learning rule can be described by the function (3):

$$\Delta w(t) = \sum_{k=1}^{n} \sum_{i_1, i_2, \ldots, i_k = 1}^{n} \left(\theta_{i_1, i_2, \ldots, i} \prod_{j=1}^{k} x_{i_j}(t-1) \right) \qquad (3)$$

where t is time, Δw is the weight change, $x_1, x_2, \ldots, x_n$ are local variables, and the θ's are real-valued coefficients, which will be determined by evolution. In other words, the evolution of learning rules in this case is equivalent to the evolution of real-valued vectors of θ's. The major aim of the evolution of learning rules is to decide these coefficients. Different θ's determine different learning rules. Due to a large number of possible terms in (3), which would make evolution very slow and impractical, only a few terms have been used in practice according to some biological or heuristic knowledge [7]. There are three major issues involved in the evolution of learning rules: a) determination of a subset of terms described in (3); b) representation of their real-valued coefficients as chromosomes; and c) the evolutionary algorithm used to evolve these chromosomes. Research related to the evolution of learning rules is also included in [25, 27], although they did not evolve learning rules explicitly. Researchers emphasized the crucial role of the environment in which the evolution occurred.

Similar to the reason explained above, also in this case there is the problem of the fitness evaluation for each individual, i.e., the encoded learning rule is very noisy because we use phenotype's fitness (a training result) to approximate genotype's fitness (a learning rule's fitness). There are two possible sources of noise. The first is the decoding process of chromosomes. The second is introduced when a decoded learning rule is evaluated by using it to train the artificial neural networks. The environmental diversity is essential in obtaining a good approximation to the fitness of the decoded learning rule and thus in reducing the noise from the second source. If a general learning rule which is applicable to a wide range of artificial neural network architectures and learning tasks is needed, the environmental diversity has to be very high, i.e., many different architectures and learning tasks have to be used in the fitness evaluation.

3 A general framework for evolutionary artificial neural networks

One distinctive feature of evolutionary artificial neural networks is their adaptability to dynamic environment, e.g. they can adapt to an environment as well as changes in the environment. In a broader sense, evolutionary artificial neural networks can be considered to be a general framework for adaptive systems, i.e., systems that can change their architectures and learning rules appropriately without human intervention. Furthermore let us focus to a description of a general framework for evolution in artificial neural networks in terms of adaptive systems where interactions among three levels of evolution are considered. This general framework can be found in figure 6 [37, 39], where different levels of evolution react to different environments and use different time scales. The evolution of connection weights proceeds at the lowest level on the fastest time scale in an environment determined by architecture, a learning rule, and learning tasks. There are, however, two alternatives to decide the level of the evolution of architectures and that of learning rules: either the evolution of architectures is at the highest level and evolution of learning rules at the lower level or vice versa. The lower the level of evolution, the faster the time scale it is on. As a rule it could be better to put the evolution of architectures at the highest level because such knowledge can be encoded in an architecture's genotypic representation to reduce the (architecture) search space and the lower-level evolution of learning rules can be biased toward this type of architectures. On the other hand, the evolution of learning rules should be at the highest level if there is more prior knowledge about them available or if there is a special interest in a certain type of learning rules. Unfortunately, there is usually little prior knowledge available about either architectures or learning rules in practice. In this case, it is more appropriate to put the evolution of architectures at the highest level since the optimality of a learning rule makes more sense when evaluated in an environment including the architecture to which learning rule is applied.

Figure 6 can also be viewed as a general framework of adaptive systems. In fact, the general framework provides a basis for comparing various specific evolutionary models of artificial neural networks according to the search procedures they used at three different levels since it defines a three dimensional space where *'0'* represents one-shot search and '$+\infty$' represents exhaustive search along each axis. Each model of artificial neural network corresponds to a point in this space.

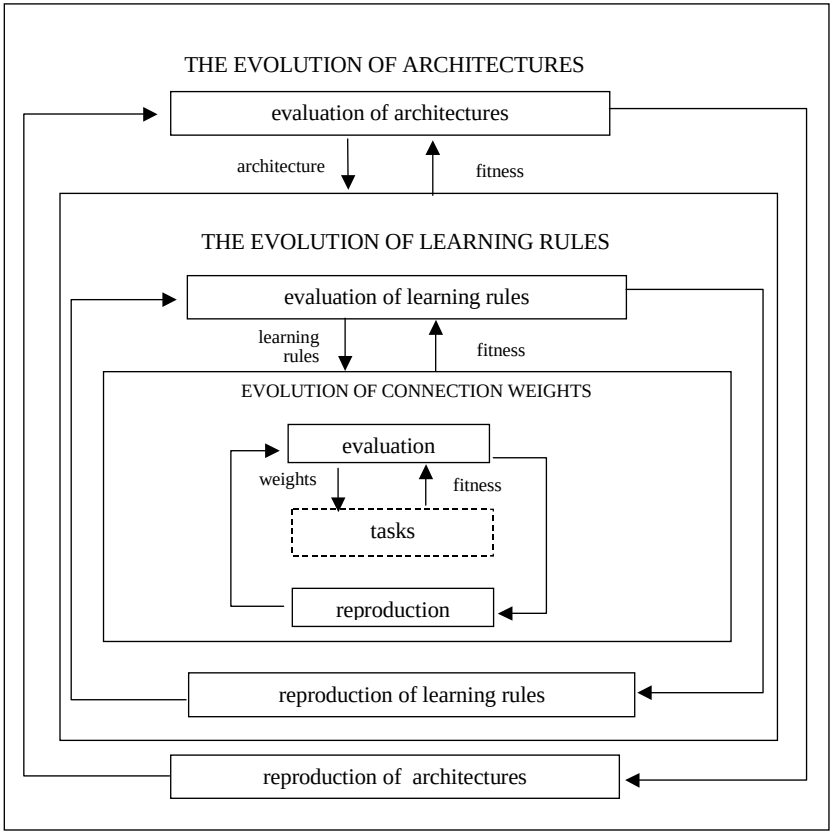

Figure 6: A general framework for evolutionary artificial neural networks.

4 Conclusions

Optimization within informatics means to seek the answer to the question "*which solution would be the best*" for a problem, in which the quality of each answer may be evaluated via a single value. Although we commonly use the word "*optimum*"; in practice we should obtain the exact global optimum within a huge complex space, which may be considered here with troubles only. Generally, solving the practical tasks, we need sufficient enough approximated (suboptimum) resolution however, above mentioned need not be implicitly a global optimum. Criterion "*sufficient enough*" differs for various types of solved problems. Evolution course

usually endeavors to find out a certain task suboptimum solution, instead of exact one.

Optimization within artificial neural networks means to seek the optimal combinations of architecture, learning rule and connection weights. 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 and other heuristic methods are very ineffective in such circumstances. Due to different time scales of different levels of evolution, it is generally agreed that global search procedures are more suitable for the evolution of architectures and that of learning rules on slow time scales, which tends to explore the search space in coarse grain (locating optimal regions), while local search procedures are more suitable for the evolution of connection weights on the fast time scale, which tends to exploit the optimal region in fine grain (finding an optimal solution). Such designed artificial neural networks have been shown to be quite competitive in terms of the quality of solutions found and the computational cost. 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.

References

[1] Angeline, P.J. “Evolving basis functions with dynamic receptive fields”. In *Proceedings of the IEEE international conference systems, man, and cybernetics, part 5 (of 5)*, pp. 4109-4114.

[2] Beale, R., and Jackson, T. *Neural Computing: An Introduction*. J W Arrowsmith Ltd, Bristol, Greit Britain 1992.

[3] Belew, R. K., McInerney, J., and Schraudolph, N. N. *Evolving networks: Using genetic algorithm with connectionist learning.* Comput. Sci. Eng. Dep. (C-014), Univ. of California, San Diego, Tech. Rep. CS90-174 (revised), Feb. 1991.

[4] Benardos, P. G. and Vosniakos, G. “Optimizing feedforward artificial neural network architecture”. *Eng. Appl. Artif. Intell.* Vol. 20 (3), 365-382. 2007.

[5] Castillo PA, Merelo JJ, Arenas MG and Romero G, “Comparing evolutionary hybrid systems for design and optimization of multilayer perceptron structure along training parameters”, *Information Sciences*, Vol 177 (14) 2884-2905, 2007.

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

[7] Chalmers, D.J. “The evolution of learning: An experiment in genetic connections”. In Touretzky, D.S., Eltman, J.L., Sejnowski, T.J., and Hinton, G.E. (eds.) *Proceedings of the 1990 Connectionist Model Summer School*. Morgan Kaufmann, San Mateo, CA 1990.

[8] Chvál, J. *Evolving Artificial Neural Networks by Means of Evolutionary Algorithms with L-Systems Based*. Encoding. Research Report, 2002.

[9] Davis, L. *Handbook of genetic algorithms*. Van Nostrand Reinhold, New York 1991.

[10] Fausett, L. V. *Fundamentals of Neural Networks*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey 1994.

[11] Ferreira, C. “Gene expression programming: A new adaptive algorithm for solving problems”. *Complex Systems*, 13 (2): 87-129, 2001.

[12] Goldberg, D. E. *Genetic algorithm in search optimalization and machine learning*. Addison-Wesley, Reading., Massachusets 1989.

[13] Gruau, F., Whitley, D., and Pyeatt, L. “A comparison between cellular encoding and direct encoding for genetic neural networks”. In Koza, J. R. Goldberg, D. E. Fogel, D. B., and Riolo, R. L. (eds.) *Genetic Programming 1996: Proceedings of the First Annual Conference*, pp. 81-89, Cambridge, MA, MIT Press, 1996.

[14] Harp, S.A., and Samad, T. “Genetic synthesis of neural network architecture”. In Davis, L. (ed.): *Handbook of genetic algorithms*, pp. 202-221. Van Nostrand Reinhold, New York 1991.
[15] Jacobs, R.A. “Increased rates of convergence through learning rate adaptation”. *Neural Networks*, vol. 1, no. 3, pp. 295–307, 1988.
[16] Kitano, H. “Designing neural networks using genetic algorithms with graph generation system”, *Complex Systems*, 4, (1990), pp. 461-476.
[17] Kvasnička, V., Pospíchal, J.,and Tiňo, P. *Evolučné algoritmy*. STU Press, Bratislava 2000.
[18] Lindenmayer, A. “Mathematical models for cellular interaction in development I, II”. *Journal of Theoretical biology* (18): 280-315 1968.
[19] Liu, B., Abbass, H.A., and McKay, B. “Classification rule discovery with ant colony optimization” In the *IEEE/WIC International Conference on Intelligent Agent Technology (IAT 2003)*, Halifax, Canada, 2003
[20] Luerssen, M. H.,. “Graph grammar encoding and evolution of automata networks”. In *Proceedings of the Twenty-Eighth Australasian Conference on Computer Science - Volume 38* (Newcastle, Australia). V. Estivill-Castro, Ed. ACM International Conference Proceeding Series, vol. 102. Australian Computer Society, Darlinghurst, Australia, 229-238, 2005.
[21] Lv, J.C. Yi, Z., and Tan, K.K. “Global Convergence of Oja’s PCA Learning Algorithm with a Non-Zero-Approaching Adaptive Learning Rate”. *Theoretical Computer Science*, vol. 367, pp. 286-307, 2006.
[22] Miikkulainen, R. “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.
[23] Miller, G.F., Todd, P.M., and Hedge, S.U. “Design neural networks using genetic algorithms”. In Schaffer, J. D. (ed.) *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 379-384. Morgan Kaufmann, San Mateo, CA 1989.
[24] Molfetas, A., Bryan, G. „Structured genetic algorithm representations for neural network evolution“. In *Proceedings of the 25th IASTED International Multi –Conference Artificial intelligence and applications*, pp 486-491, Austria 2007.
[25] Nolfi, S., Elman, J. L., and Parisi, D. *Learning and evolution in neural networks.* Center Res. Language, Univ. California, San Diego, July Tech. Rep. CRT-9019, 1990.
[26] Nolfi S., Parisi D., „Evolution of artificial neural networks”. In M. A.Arbib (Ed.), *Handbook of brain theory and neural networks*, Second Edition. Cambridge, MA: MIT Press, pp 418-421, 2002.

[27] Parisi, D., Cecconi, F., and Nolfi, S. “Econets: neural networks that learn in an environment”. *Network*, 1:119-168, 1990.
[28] Rossi, F., and Conan-Guez, B. “Functional multi-layer perceptron: a non-linear tool for functional data analysis”. *Neural Networks.* Vol 18 (1) 45-60, 2005
[29] Shafi K., Abbass H. and Zhu W. “Real time signature extraction during adaptive rule discovery using UCS”, In *IEEE Congress on Evolutionary Computation (CEC),* Singapore, 25-28 September, 2007
[30] Stanley, K.O., and Miikkulainen, R. “Evolving adaptive neural networks with and without adaptive synapses”. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002) Workshop Program*. San Francisco, CA: Morgan Kaufmann, 2002.
[31] Stanley K.O. *Efficient evolution of neural networks through complexification.* Ph.D. dissertation, University of Texas at Austin, 2004.
[32] Stanley, K. O. “Comparing Artificial Phenotypes with Natural Biological Patterns”. *In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) Workshop Program.* New York, NY: ACM Press, 2006.
[33] 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)** (1998), pp. 653-664. ISSN: 1210-0552.
[34] Volná, E. „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. ISBN: 978-972-8865-86-3.
[35] Volná, E. "Designing Modular Artificial Neural Network through Evolution". *In* J. Marques de Sá, L. A. Alexandre, W. Duch, and D.P.Mandic (eds.) Artificial Neural Networks – ICANN’07, Lecture Notes in Computer Science, vol. 4668, Springer-Verlag series, 2007, pp 299-308. ISBN 83-919932-4-8. ISSN 0302-9743.
[36] Xu, R., Wunsch, D. “Survey of Clustering Algorithms”, *IEEE Transactions on Neural Networks*, Vol. 16(3) 645-678, 2005.
[37] Yao, X., “Evolutionary artificial neural networks.” In Kent, A., and Williams, J. G. (eds.) *Encyclopedia of Computer Science and Technology*, vol. 33, New York: Marcel Dekker, 1995, pp. 137–170.
[38] Yao, X., Lin, G., and Liu, Y. “A new evolutionary system for evolving artificial neural networks”. *IEEE Trans. Neural Networks*, vol. 8, pp. 694–713, 1997.
[39] Yao, X. “Evolving artificial neural networks”. In *Proceedings of the IEEE* 89 (9) 1423-1447, 1999.

[40] Zhang, J., Chung, H. S., and Zhong, J. 2005. Adaptive crossover and mutation in genetic algorithms based on clustering technique. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation* (Washington DC, USA, June 25 - 29, 2005). H. Beyer, Ed. GECCO '05. ACM, New York, NY, 1577-1578.

RNDr. PaedDr. Eva Volná, PhD.

Curriculum Vitae

Education:

MSc, (PaedDr., with honor) in Mathematics and Physics Teaching, Pedagogical Faculty of Ostrava (*at present* University of Ostrava), Ostrava, 1985

MSc, (RNDr.) in Theoretical Cybernetics, Mathematical Informatics, and Theory of Systems, Faculty of Science, Jan Evangelista Purkyně University in Brno (*at present* Masaryk University), Brno, 1991

PhD. (Ph.D.) in branch 25-11-9 Applied Informatics, Slovak University of Technology in Bratislava (STU), Bratislava, Slovakia, 2003 Thesis: Modular Neural Networks

Employment:

1985-1986	Study stay, Academy of Science, Ostrava, CZ
1986-1992	Scientific and research worker, Academy of Science, Ostrava, CZ
Since 1992	Assistant professor, Department of Informatics and Computers, Faculty of Csience, University of Ostrava, Ostrava, CZ

Pedagogical activity:

Neural Networks I (since 1994), Neural Networks II (since 1998), Selected Aspects of Artificial Intelligence (since 2002), Artificial Intelligence (since 2004)

1998 -2008	Preparation of 8 educational texts.
1998 -2008	Supervision of approx. 30 bachelor and diploma thesis that were successfully defended.

Research experience:

Neural Networks, Evolutionary algorithms, Pattern Recognition, Artificial Intelligence.

Publications summary:

More than 50 papers in technical journals and proceedings of conferences.