A Taxonomy of the Evolution of Artificial Neural Systems

Helmut A. Mayer *

Abstract. *Biological neural networks having been shaped in billions of years of evolution are the source of numerous, complex capabilities of living organisms. Especially, the form of intelligence attributed to humans has inspired computer scientists to model the evolution of neural networks with the ultimate goal to create computer systems with cognitive abilities. However, considering the current state of research in the domain of artificial neural network (ANN) evolution there is still a huge gap between the extremely versatile brains of higher organisms, and the very specialized artificial neural systems mostly applied to a single, well–defined task.*

The majority of work on ANN evolution is concerned with the optimization of network structure and/or network weights so as to achieve maximal network performance for a specific problem. We present a taxonomy in order to categorize the (co)evolution of various components of an artificial neural system (ANS). We illustrate some of the approaches in ANS evolution by a few examples from the literature, and present our extensions to evolution of the ANN learning component by means of experiments with the two spirals benchmark problem utilizing the netGEN system. Specifically, we demonstrate that the evolution of non–monotonous activation functions of (nearly) arbitrary shape can enhance the performance of an ANS.

1. Introduction

The history of mankind is tightly coupled with the history of tools invented and used by our ancestors. Tools, and in recent history machines, extend the physical capabilities of humans. With the advent of computers in the first half of the 20th century the era of machines extending the intellectual capabilities of humans has begun. Since then, one of the most fascinating goals in computer science is to add "intelligence" to computers that until today are believed to operate in a strict mechanistic fashion. In simple words, a computer does exactly what it was programmed to do. Although, there is no consensus on the definition of intelligence, e.g., it may be argued that a conventional chess program beating a human is more intelligent than its opponent, there is agreement that computers have no or only very restricted learning capabilities.

Learning, however, is a prerequisite for human intelligence, and if we want to model (and extend) the cognitive capabilities of humans, the machine must be able to learn. The most natural among various concepts to achieve learning in computers is to model the fundamental properties of human learning. In a general form learning is equivalent to controlled changes in the complex neural circuitry of the brain. In the artificial model the control mechanisms governing the adaptation of neural parameters

*Department of Scientific Computing, University of Salzburg, AUSTRIA, email: helmut@cosy.sbg.ac.at

are built into training algorithms being an important area of research in artificial neural networks. Obviously, connectionist learning demands a given neural structure, i.e., a brain, which has been honed in billions of years of evolution.

Consequently, researchers also transferred the concepts of natural evolution to computer science using the basic principle of "survival of the fittest" for the evolution of artificial neural structures and optimization problems in general. By modeling brain evolution the old philosophical discussion of the distribution of inherited (phylogenetic learning) and acquired (ontogenetic learning) knowledge also arises in the artificial domain. This central question has led to a variety of approaches, where not only the neural (physical) structures are evolved, but also other aspects of an artificial neural network, e.g., the learning scheme, or the structure and composition of training patterns.

Realizing that the final performance of an artificial neural network is not only dependent on its structure, but also on other components with great influence, we will use the term "artificial neural system". In the following we identify the single components of the system and those of the evolutionary methods employed to construct an artificial neural system without human intervention.

2. Taxonomy of ANS Evolution

Today *Evolutionary Computation* (EC) methods are used for optimization of a variety of components of neural systems. In order to categorize the various components we will present a taxonomy of the evolutionary optimization of an *Artificial Neural System* (ANS) [20]. To our knowledge, the first work presenting a survey of the various aspects of the evolution of neural networks was given by Yao (1993). In the latter work the field of *Evolutionary Artificial Neural Networks* (EANNs) has been divided into three categories: the evolution of weights, the evolution of architecture, and the evolution of learning rules [32]. This taxonomy has been extended and refined by adding the categories evolution of input neurons, and evolution of neural network ensembles in [33].

Another taxonomy of EANNs has been suggested by Balakrishnan and Honavar (1995). Here the evolutionary design of neural architectures is grouped into four main categories: the genotype encoding scheme, the ANN type, the variables of evolution, and the application domain [1]. Naturally, these attempts to structuralize the field of EANNs overlap, e.g., Yao (1999) presented the encoding scheme as a sub–category of the evolution of architecture, but we not only want to merge and unify these taxonomies, but put it in a broader context, namely, the evolutionary optimization of ANSs.

We define an ANS to be any hardware or software system whose internal model of computation is based on distributed, sub–symbolic information processing as believed to be the fundament of the operation of *Biological Neural Networks* (BNNs). But an ANS not only consists of the intrinsic components of an ANN (neurons, connections, weights, bias, activation functions), but also of its "environment", e.g., training data sets being an integral component of an ANS, if the ANN is to be trained.

In fact natural evolution exploited the existence of a (cultural) environment, when adapting and enlarging the brains of primitive life forms exhibiting an imprinted stimulus–response behavior. Over billions of years the addition of more and more complex brain structures culminated in the *Neo–cortex* of humans which can be used for a broad variety of cognitive tasks. The universal applicability comes at the price of years and decades of learning, as all of us have (sometimes painfully) experienced. Evidently, learning can only sculpture our brains in a meaningful way, if there is a cultural environment,

i.e., parents, teachers, media.

Consequently, we would like to present a possible taxonomy of an ANS whose components may be optimized by evolution (or any other optimization technique being applicable). Hence, we separate ANS components and the methods utilized to construct them, and start by identifying the main components of an ANS (Figure 1).

- ANN Structure

  We define the structure of an ANN as its "physical" layout, i.e., the graph representing the network. The vertices V of the graph represent the neurons, while the edges E of the graph represent the connections. In the most general form any graph can be evolved. Hence, the number of neurons and connections, and the specific connectivity is subjected to evolution. For specific ANN types certain restrictions have to be imposed on the graph, e.g., the avoidance of cycles when searching for feed–forward networks. The neurons and connections are not further specified within the structure component.

- ANN Learning

  From a global perspective learning in ANNs is equivalent to adjusting its internal parameters, e.g., weights and biases. It should be noted that we could even view the structure as a parameter being adjusted by learning which may be even the most important aspect of learning in BNNs. However, most conventional ANN training methods do not change ANN structure. Therefore, we think that the discrimination of structure and learning method is a legitimate one for ANSs. This component encompasses the evolution of network parameters, learning parameters, and learning rules, as all these directly or indirectly adjust the internal parameters of an ANN.

  Evolution of network parameters, e.g., weights, biases, and activation functions, is the most direct approach to ANN learning, hence it could be termed *Evolutionary Training* of ANNs. A more indirect approach utilizing the impressive work on ANN training based on *Error Gradient Descent* is the evolution of learning parameters of a given learning algorithm. The learning parameters not directly change the network parameters, but alter the behavior of the learning algorithm. Finally, evolution of learning rules is an even more indirect approach, as the training algorithm itself is evolved. With the latter approach not the behavior of a single training algorithm is altered by evolution, but a variety of learning algorithms is sampled and shaped by evolutionary processes.

- ANN Teaching

  For humans it is evident that a good learning technique (training algorithm) is of little worth, if there is no appropriate learning material (training data) or teacher. However, the importance of the latter is not reflected in the scientific work dealing with ANSs. Training, validation, and test data are mostly considered as a given entity, and rarely the question is raised, if these data are sufficient to train or evaluate an ANN. Though, non–supervised training methods generate ANNs without the need of training data, the performance of the network is still measured using test data the ANN has never "seen" before. Again, the question remains, if any given test data are a reliable indicator of ANN performance.

  In order to improve this situation we could generate data sets for training, validation, and testing by simply evolving data subsets of all available data. We could also evolve data "from scratch" within restrictions given by the specific task the ANN is used for. A very elegant way to get

rid of arbitrary human choices of the training and testing environment is coevolution. In the most natural setting the testing environment is another ANN as is the case in coevolution of game–playing ANNs. Here, the test data (or better test scenarios) for an ANN are simply the decisions of an ANN opponent. This self–playing of ANNs implicitely achieves evolution of the teaching component by evolution of the learning component (and vice versa).

- ANN I/O Representation

  Certainly the least explored evolvable ANS component is the mapping of the internal state (neural activation) of an ANN to the specific meaning of that state. In less technical terms it is the question of ANN input and output representation. Traditionally, the human ANN designer defines the number of input/output neurons, and the representation (coding) of input/output values. Whereas the number of input/output neurons can be evolved via evolution of ANN structure, implicitely performing feature selection at the input layer, the representation (coding) of input and output values could be evolved separately. To our knowledge few researchers have started to work in this direction.

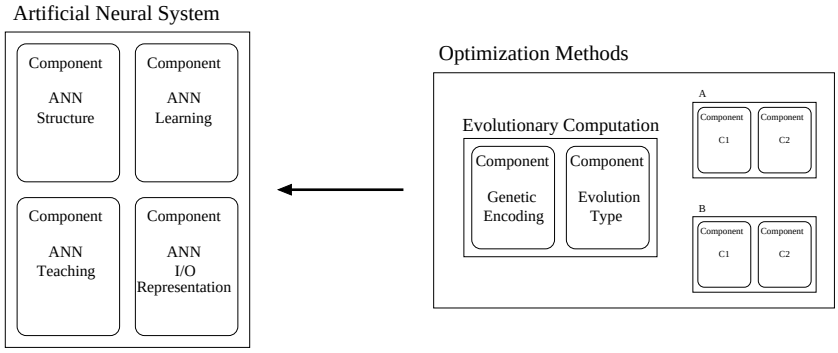

Figure 1. System view of the optimization of ANS components.

A variety of optimization techniques has been applied to assist in the (semi)–automatic construction of above ANS components. The general method of choice presented in this work to achieve this goal is evolutionary computation. Identifying the key components of an *Evolutionary Algorithm* (EA) for ANS construction, we only consider basic design issues common to all EA instances. When speaking of individual ANNs and their environment the term *Coevolution* readily comes to mind. Not surprisingly, work on evolution of ANSs has adopted this paradigm for optimization of ANS components [26, 25, 19, 5]. In the following the basic EA components for automatic construction of an ANS are given:

- Genetic Encoding

  In order to put the above ANS components within the framework of EAs the human designer has to devise an appropriate encoding scheme. The genetic encoding can be grouped into *Direct* and *Indirect* encoding. With direct encoding of an ANS component (or any evolvable component of a system) the component is completely described by the parameter values extracted from the genotype, while indirect encoding uses the genetic information as input to an algorithm (decoder) which finally generates the parameter values describing the ANS component. E.g., the ANN training component can be evolved directly (evolution of weights and biases) or indirectly (evolution of learning parameters for a training algorithm).

The evolved parameters can be encoded as a bitstring, or a vector of real numbers [1], which in turn influences the design of mutation and recombination operators. Over the last years the borders between the different evolutionary techniques have more and more vanished, thus the term *Evolutionary Algorithm* (EA) should be used describing a method based on artificial evolution. Specifically, when evolving ANS components, the blending of techniques becomes obvious, as an ANN could be evolved by encoding the structure with a bitstring and the weights with real numbers.

- Evolution Type

  The standard approach to evolve an ANS is to encode the components, e.g., ANN structure and ANN teaching, on a single genotype representing the complete ANS. A single population of these individuals is then subjected to selection, mutation, and recombination. The fitness is determined by the performance of the network on test data or test scenarios. Usually, the test environment is static, i.e., the same fitness function is applied to each individual in the evolutionary run, hence we would like to call this type of evolution *ANS Evolution*.

  If ANS components are evolved in specific populations, each resembling one or more components, we speak of *ANS Coevolution*. In this case the environment of an ANS component, e.g., the training data set of an ANN structure, is also subjected to evolution, hence it is non–static. Coevolution may be cooperative (symbiotic) or competitive (parasitic). The fitness of a population cooperating with another is based on the joint success of the populations, while a population competing with another is inversely proportional to the fitness of the (hostile) population.

  It should be stressed that we use the term coevolution only for processes at the level of populations (a number of individuals). It could be argued that coevolution also takes place within a single individual, specifically, at the genotype level. Considering a very simple optimization problems with two parameters $x, y \in \mathbb{R}$ encoded on an artificial chromosome, only in the case of a linear problem the two parameters can be optimized separately. In the very common case of a non–linear (*epistatic*) problem the two parameters have to "cooperate" to achieve a good fitness (solution). The search for these parameters by an EA could also be labeled coevolutionary, but for the sake of a clearer separation of meanings we will use the term coevolution as explained above.

In order to fill some of the general ANS components with life, we take a closer look at the most investigated components ANN structure and ANN learning.

3. Evolution of ANS Components

ANNs have been successfully applied to a wide variety of problems in science, business, and engineering, but an analytical rule for the construction of a variety of ANS components for a given problem has not been discovered yet. In this section we discuss the evolution of those ANS components in more detail, which are of specific importance for the experiment utilizing the netGEN system in Section 4.

[1] Most researchers use the term *real* number, though only *rational* numbers can be stored and processed by a digital computer.

3.1. Evolution of ANN Structure

The search for a problem–adapted ANN structure is a key issue in ANN research. The complexity of this task is underlined by the fact that the space of possible network structures is enormous, as there are $2^{\frac{n(n-1)}{2}}$ ways to connect n neurons, if the nodes are labeled (distinguishable), all partially connected nets are counted, and self–connections are excluded. Even for a very moderate network size of $n = 20$ this results in 1.57×10^{57} different networks.

ANN structure has direct impact on training time, convergence and generalization ability. The complexity of the structure increases with the number of neurons, the number of connections, and the connectivity patterns. It has been found that ANNs with lower complexity, i.e., small number of neurons and/or sparse connectivity, show a better generalization performance than more complex networks [30] which tend to lose generalization ability due to *overfitting*.

Various non–evolutionary methods have been suggested for the automatic construction of ANN structure. These algorithmic approaches can be divided into *Network Growing* and *Network Pruning*. Network growing [18] starts with a minimal structure and iteratively adds neurons to the network as long as specific performance measures increase, e.g., *Cascade Correlation* [7]. On the other hand network pruning [27] starts out with a maximal structure and deletes neurons and/or connections based on various heuristics, e.g., *Optimal Brain Damage* [6], *Optimal Brain Surgeon* [11].

In contrast to above sequential algorithms the evolutionary search for ANN structures [32, 2] is a parallel search, as complete ANN structures are evolved and evaluated based on the performance of the trained ANN structure. Besides the definition of the fitness function, the other main design issue in all applications of EAs is the genetic encoding of an individual (solution). As pointed out above the genetic encoding can be direct [22, 31], or indirect [16, 9, 8, 12].

3.2. Evolution of ANN Learning

The main reason for the wide applicability of ANNs is their generic learning capability. Learning in ANNs can always be reduced to numerical changes of internal parameters, most noteably, the weights of connections. Adaptation of these internal parameters changes the activation of neurons including the output neurons, which usually represent the response (answer) to a certain input (question). In order to change the internal parameters in a way that allows the network to generate (nearly) correct outputs to given inputs a variety of training methods have been devised. Many of these training methods demand training data in the form of known input/output patterns, which are used to tune the network so as to generate the desired output to a given input pattern. Generally, all these methods try to minimize ANN error measured as a distance of actual network output to the desired output. Analytical training methods, most prominently *Gradient Descent* approaches have been successfully applied in a broad variety of ANSs, but certain requirements have to be met in order to be able to use these analytical methods.

These training algorithms demand differentiable activation functions, a given network structure, and a set of training patterns. While the first two prerequisites can be met by using standard ANNs, the latter is dependent on the specific problem. Sometimes it is difficult, if not impossible, to obtain training patterns of sufficient quantity and/or quality, hence non–analytical training methods have to be used. If there is a way to assess the overall performance of the net, evolutionary computation techniques are a promising candidate to train an ANN. The parameters of the network can be encoded in a genotype,

which is mapped into a phenotypic ANN, and the fitness is based on the overall "behavior" of the network. But even, when training data are available, evolutionary ANN training could be beneficial, as it performs a less local search than gradient descent methods which easily get stuck at local optima.

Though, evolutionary training has the potential to escape from local optima, it should not be overlooked that the search space is enormous. Again, the fact that most ANN applications employ very small networks alleviates the problem. But even in a modest network with 100 connections, evolution faces an infinite, 100–dimensional space (limited to a finite space in computers), when searching for weights. As a consequence, the transition from a direct encoding [24, 35] of network parameters to indirect encoding approaches[10, 3, 15, 14] may be the only choice to be able to deal with increasing complexity.

Basically, indirect encoding of ANN learning can be divided into two branches. The parameters of analytical training algorithms can be evolved, hereby decreasing complexity, as the search in weight space is confined to the paths governed by the training algorithm. However, the rich empirical evidence of successful applications of well–known training methods ensures that a good solution will be found. More general, but still indirect, is the evolution of learning rules, which can be loosely compared to *Genetic Programming* (GP) approaches [17], where algorithms are evolved. In the context of ANSs the evolved program is the learning algorithm, which controls the change of internal parameters.

4. netGEN Experiments

Having described more theoretical aspects of the evolutionary optimization of ANS components, now we would like to present some experiments with the netGEN system. We use synthetic benchmark data describing the *TwoSpirals* problem, a dichotomous classification task that is known to be hard to solve for ANNs. We pay specific attention to the evolution of ANN teaching, namely, the evolution of non–monotonous activation function (AF) templates [21].

An AF template is a definition of an activation function being assigned to specific neurons in the network. Both, AF templates and their distribution to all neurons in the network are evolved. This may be seen as a loose analogy to neuron differentiation in biological neural networks (BNNs). While BNNs employ different neuron types in functionally different brain areas, neuron differentiation in ANNs might be useful to increase the adaptability to specific problems. The evolution of AF templates is based on evolving the control points of a cubic spline function, hence, non–monotonous AFs of (nearly) arbitrary shape may be generated.

4.1. Evolution of ANN Structure and Activation Function Templates

The technical platform for the evolution of ANNs is the netGEN system searching a problem–adapted ANN architecture by means of an *Evolutionary Algorithm* (EA), and training the evolved networks using the *Stuttgart Neural Network Simulator* (SNNS) [34]. In order to speed up the evolutionary process, an arbitrary number of workstations may be employed to train individual ANNs in parallel [13]. The ANN's genetic blueprint is based on a direct encoding suggested by Miller et al. [22]. The basic structure of the genotype is depicted in Figure 2.

The binary encoded ANN chromosome contains four main sections. The *Learn Parameters* encode values used for ANN training, the *Activation Function Template Parameters* are used to describe

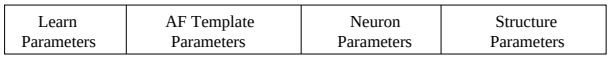

Figure 2. The organization of the ANN genotype.

one or more activation functions, the *Neuron Parameters* indicate the neuron type, and the *Structure Parameters* explicitely specify each connection of the network (direct encoding). Figure 3 shows the details of the learn parameter and the AF template parameter section.

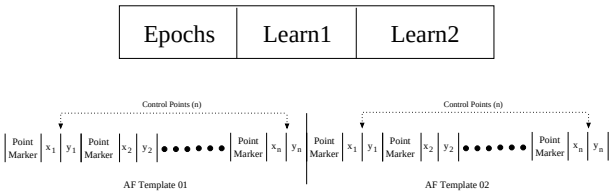

Figure 3. Details of the learn parameter section (above) and the activation function template parameter section (below) of the ANN genotype.

The learn parameter section contains the number of epochs and two learning parameters for the selected training algorithm *Resilient Back–propagation* (RProp) [28]. The evolution of the number of epochs for ANN training is a measure to avoid overfitting with the additional benefit of another critical ANN parameter being discovered by evolution.

The most interesting feature of the presented encoding are *Markers* – single bases (bits) which are a simple analogue to activators/repressors regulating the expression of wild–type genes. A hidden neuron marker (part of the neuron parameters) determines, if the specific neuron associated with it is present in the decoded network. As the number of output neurons is usually strictly dependent on the specific problem the ANN should learn, these neurons are not controlled by markers.

The activation function template is composed by a maximum number of n control points of a cubic spline. In order to also allow fewer control points, each control point is "regulated" by a point marker. If a point marker indicates the absence of the corresponding point, the control point is not used for cubic spline calculations to construct the AF template. Similarly, an active neuron marker prunes the neuron it controls, and all connections associated with that neuron become obsolete. As a consequence, most ANN chromosomes contain noncoding regions [19]. When using the single logistic AF template, only the learn and structure parameters are evolved. If a neuron is expressed, the neuron type is given by an AF index in the neuron parameters. Evidently, when a single AF template is evolved the size of the bit field encoding the index is 0.

ANN structure is represented by a linearized binary adjacency matrix. As in this work we are only concerned with feed–forward architectures, all elements of the upper triangle matrix must be 0, hence they are not included in the ANN genotype. Due to this linearization we are able to use the standard 2–point crossover operator for recombination. When constructing the ANN phenotype the AF index (0, if neuron not expressed) is stored in the main diagonal of the connection matrix as shown in Fig. 4 for an exemplary network.

The maximum number of hidden neurons (neuron markers) has to be set in advance with this encoding scheme, hence, it could be labeled as *Evolutionary Pruning*, since the system imposes an upper bound on the complexity of the network.

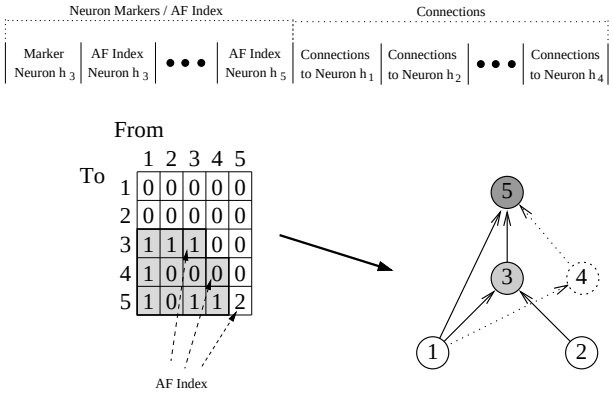

Figure 4. Genotype/Phenotype mapping of ANN structure.

A. *ANN Fitness Function*

The ANN fitness function comprises a complexity regularization term $\mathcal{E}_c = |C_{total}|$, with C_{total} being the set of all network connections. The *Composite Fitness Function* $\mathcal{F}$ is given by a weighted sum of *Model Fitness* and *Complexity Fitness*

$$\mathcal{F} = \alpha_1 \frac{1}{1+\mathcal{E}_m} + \alpha_2 \frac{1}{1+\mathcal{E}_c} \tag{1}$$

with $\alpha_1 + \alpha_2 = 1.0$, and $\mathcal{E}_m$ being the model error. Earlier work showed that α_2 in the range of $0.001 - 0.01$ is sufficient to guide the evolution towards ANNs of low complexity. In effect the complexity term acts as a "tie–breaker" between different ANNs with (nearly) identical model error, where the less complex network receives a slightly better fitness. In this work we set $\alpha_1 = 0.99$.

B. *EA and ANN Parameters*

The following EA and ANN parameters have been used with all the experiments in this paper:

EA Parameters: Population Size = 50, Generations = 50, Crossover Probability p_c = 0.6, Mutation Probability p_m = 0.005, Crossover = 2–Point, Selection Method = Binary Tournament.

ANN Parameters: Network Topology = Generalized Multi–Layer Perceptron, Activation Function (hidden and output neurons) = Sigmoid or evolved Cubic Splines, Output Function (all neurons) = Identity, Training = RProp, Learning Parameters Δ_0 (max 1.0), Δ_{max} (max 50.0), Number of Training Epochs (max 1000) = Evolutionary.

C. *Cubic Spline Parameters*

A cubic spline activation function is described by n control points $(x_i, y_i) \in \mathbb{R}\times\mathbb{R}$ where $i = 1, \ldots, n$. These n points define $n-1$ intervalls on the x-axis denoted by $x \in [x_i, x_{i+1}]$ with $x_{i+1} > x_i$. In each of these intervalls a function $f_i(x)$ is defined by

$$f_i(x) = f_i(x_i) + a_i(x - x_i) + b_i(x - x_i)^2 + c_i(x - x_i)^3 \quad (2)$$

with constants $a_i, b_i, c_i \in \mathbb{R}$. Demanding equality of the function value, and the first and second derivative at the interval borders the constants can be determined for each interval yielding a continuous and differentiable function composed of a number of cubic splines.

With respect to the ANN genotype (Fig. 3) the number of templates n_T, and the maximal number of control points n_c has to be set prior to the evolutionary run. Also, the x–, and y–range of the cubic spline activation function has to be fixed to specific intervals $[x_{min}, x_{max}]$ (sensitivity interval) and $[a_{min}, a_{max}]$ (activation interval), respectively. However, within these bounds the evolutionary process may freely float. For all experiments we set the number of control points $n_c = 8$ and the activation interval $[a_{min}, a_{max}] = [0.0, 1.0]$. The sensitivity interval depends on the specific problem.

4.2. Experimental Results

The main question to be adressed is the performance of a cubic spline network in comparison to an ANN with conventional, sigmoidal AFs. Intuitively, the decision boundaries in classification problems could be modeled more easily by cubic splines of great plasticity than by the "frozen" logistic function. Thus, we choose the synthetic, but nevertheless hard *Two Spirals* problem with complex, nonlinear decision boundaries.

The task in the two spirals problem is to discriminate between two sets of points which lie on two distinct spirals in the plane. These spirals coil three times around the origin and around one another [4]. Training, validation, and test set comprise 194 patterns each. The basic ANN structure is defined by two input neurons and two output neurons (one for each spiral, winner takes all).

During the evolutionary process the ANN structure is trained on a training set and evaluated on a validation set (ANN fitness). Finally, the performance of the evolved best net is measured on a test set. Each evolutionary run is repeated 20 times. For this classification problem the model error in the fitness function (Equation 1) is simply given by $\mathcal{E}_m = \frac{e_v}{n_v}$, where e_v is the number of misclassifications on the validation set, and n_v its respective size.

After having extended the netGEN system with the evolution of AF templates, we performed some runs with the simple XOR problem so as to test the new software components. Surprisingly, when evolving AF templates ($n_T = 1$, sensitivity interval $[-10.0, 10.0]$) for the XOR net, we mostly arrived at networks without any hidden neurons (two inputs, one output with cubic spline AF). This seems to contradict the well–known fact that a single perceptron cannot learn the XOR function known as *Minsky's Paradox* [23]. In a strict sense our evolved XOR net is not a perceptron which is defined having a threshold AF, but even a logistic AF does not do the trick. While these functions have the desirable non–linearity, they are monotonous in contrast to the non–monotonous evolved cubic spline AFs. A non–monotonous triangle AF being a simplification of the evolved cubic spline AFs in our experiments enables a single neuron to solve the XOR problem. Rueda and Oommen (2000) reported on this observation in the context of statistical pattern recognition proposing pairwise linear classifiers [29].

The details of two evolved XOR nets are shown in Fig. 5.

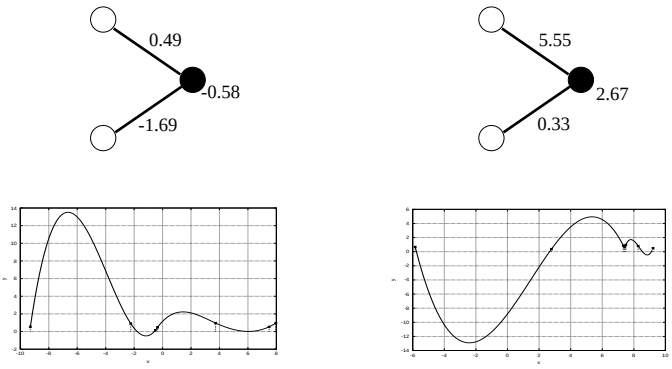

Figure 5. Evolved XOR nets with weights, bias, and cubic spline activation function.

For experiments with the two spirals problem the sensitivity interval has been set to $[-100.0, 100.0]$. The results for different numbers of AF templates ($n_T = 1, 2, 4$) are presented in Table 1. For comparisons the results of the evolution of an ANN structure with sigmoid AF ($n_T = 0$) are included in Table 1.

TwoSpirals					
Templates	Structure		Test Set		
	#H	#C	Acc	StdDev	Best
$n_T = 0$	14.75	125.00	0.8320	0.0301	0.8969
$n_T = 1$	13.10	89.30	0.9327	0.0218	0.9639
$n_T = 2$	12.84	85.90	0.9235	0.0260	0.9639
$n_T = 4$	12.30	81.45	0.8768	0.0646	0.9536

Table 1. ANN performance and structure of evolved ANNs with different numbers of evolved AF templates (with complexity term, averaged on 20 runs).

It can be seen that the performance of the spline networks is considerably superior to the sigmoid networks. Though, in both cases ANN structure has been evolved using the complexity term, the spline networks are clearly less complex than the sigmoid networks. These results are perfectly in line with our initial hypothesis that neurons with AFs adapted to the problem structure are of increased computational power. E.g., a rectangular function can be approximated easily by a single neuron with a rectangular AF, but needs a larger number of sigmoid neurons for a similar approximation error.

The decrease in performance with increasing number of AF templates mainly stems from the growing complexity of the genotype. In our experiments a single AF template is encoded by eight control points consisting of $x-$ and y coordinates, where each coordinate uses eight bases of genetic code. With the additional point markers (Figure 3) this results in a total of 136 bases to encode an AF template. As the evolutionary time given to networks with varying numbers of AF templates is kept constant (50 individuals, 50 generations), it seems reasonable that more complex genotypes proliferate ANNs with weaker performance.

It should be noted that the exact a priori knowledge of the (circular) decision boundaries of the two spirals problem makes it an excellent candidate for networks with AFs other than the sigmoid function. This assumption is confirmed by the receptive fields of the sigmoid and a spline network shown in Figure 6 (for esthetic reasons the discrete points of the test set are joined to form a continuos spiral).

The receptive field is generated by sampling the unit square with a small step width and recording the network's classification for each point.

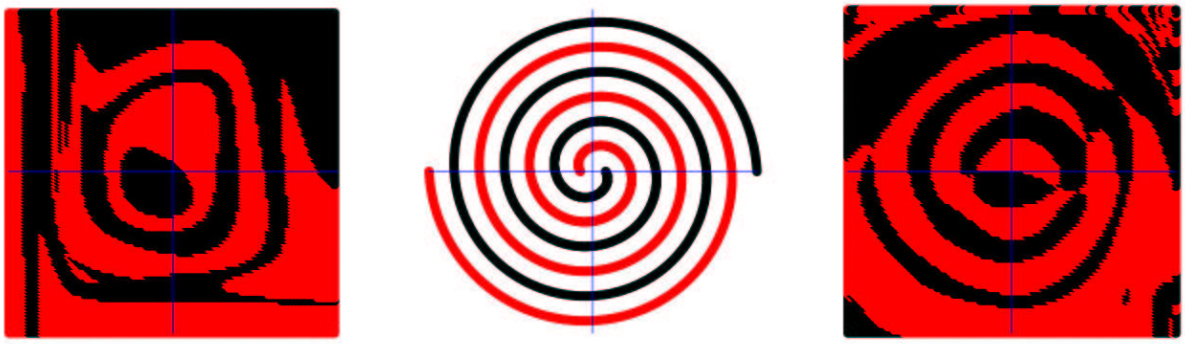

Figure 6. Receptive fields of the best evolved sigmoid ANN (left) and best evolved spline ANN ($n_T = 2$, right), and the test set (middle).

The evolved sigmoid ANN generates pronounced non–contiguous subspaces for the two classes which make misclassifications inevitable. The shape of the decision boundaries of the evolved spline net much more resemble the two spirals. There are only two small spots, where the light spiral (middle of upper right quadrant) and the dark spiral (left end of x–axis) are disrupted. The above results underline the potential of evolved cubic spline activation functions including non–monotonous functions.

Although, the non–linear activation function is at the core of the specific characteristics of an ANN, the AF's importance is not reflected in the work on evolution of ANSs. We hope to be able to stipulate more work in this promising direction in the future.

5. Summary

We have presented a taxonomy of evolutionary optimization of artificial neural system (ANS) components. We have identified the ANS components ANN structure, learning, teaching, and I/O representation. Out of a variety of existing methods to automatize the process of ANS construction, we employ the evolutionary computation paradigm modeling the natural process of Darwinian selection. In a specific application example we demonstrated the evolution of the ANS components ANN structure and ANN learning for the two spirals benchmark problem.

Evolution of the ANS learning component can be achieved in a variety of ways ranging from evolution of network weights using evolution strategies to evolution of training algorithms with a genetic programming approach. Though, the ANN activation function (AF) is ultimately responsible for the nonlinear properties of a network definitely contributing to the learning capabilities of an ANS, AFs are rarely viewed as free network parameters. By evolving cubic spline AFs of (nearly) arbitrary shape we could observe considerable improvements, when comparing classification results of ANNs with conventional, sigmoidal AFs to networks containing evolved (non–monotonous) AFs. By inspecting the decision boundaries of the two types of networks the potential of the spline nets to model nonlinear boundaries of class subspaces with great accuracy have been demonstrated.

As ANS evolution is a generic technique applicable to a wide range of problems, the netGEN system (and its successor the Java framework netJEN) is the starting point of some challenging research directions we are currently pursuing. Besides continuing work to improve specific netJEN components, among them the evolution of cubice spline AFs, and the multi–chromosomal encoding of an ANS, we are employing netJEN for the coevolution of Go playing ANNs, and the evolution of robotic neurocontrollers trained by reinforcement learning methods.

References

[1] Karthik Balakrishnan and Vasant Honavar. Evolutionary design of neural architectures – a preliminary taxonomy and guide to literature. Technical Report CS TR #95-01, Iowa State University, Department of Computer Science, Ames, Iowa 50011–1040, U.S.A., January 1995.

[2] Jürgen Branke. Evolutionary Algorithms in Neural Network Design and Training – A Review. In Jarmo T. Alander, editor, *Proceedings of the First Nordic Workshop on Genetic Algorithms and their Applications*, pages 145–163, 1995.

[3] David J. Chalmers. The Evolution of Learning: An Experiment in Genetic Connectionism. In D. S. Touretsky, J. L. Elman, T. J Sejnowski, and G. E. Hinton, editors, *Proceedings of the 1990 Connectionist Summer School*, pages 81–90, San Francisco, CA, 1990. Morgan Kaufmann.

[4] CMU-Repository. ftp://ftp.cs.cmu.edu/afs/cs.cmu.edu/ project/connect/bench/. WWW Repository, Carnegie Mellon University, 1993.

[5] André Coelho, Daniel Weingaertner, and Fernando J. von Zuben. Evolving Heteregeneous Neural Networks for Classification Problems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 266–273, San Francisco, July 2001. Morgan Kaufmann.

[6] Y.L. Le Cun, J.S. Denker, and S.A. Solla. Optimal Brain Damage. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 599–605. Morgan Kaufmann, 1990.

[7] S.E. Fahlman and C. Lebiere. The Cascade-Correlation Learning Architecture. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 524–532. Morgan Kaufmann, 1990.

[8] Christoph M. Friedrich and Claudio Moraga. An Evolutionary Method to Find Good Building Blocks for Architectures of Artificial Neural Networks. In *Proceedings of the Sixth International Conference on Information Processing and Management of Uncertainty in Knowledge–Based Systems*, pages 951–956, 1996.

[9] F. Gruau. Genetic Synthesis of Boolean Neural networks with a Cell Rewriting Developmental Process. In D. Whitley and J. David Schaffer, editors, *Proceedings of the Third International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 55–74, Los Alamitos, California, 1992. IEEE Computer Society Press.

[10] Steven Alex Harp, Tariq Samad, and Aloke Guha. Towards the genetic synthesis of neural networks. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 360–369, San Mateo, California, 1989. Philips Laboratories, Morgan Kaufmann.

[11] B. Hassibi, D.G. Stork, and G.J. Wolff. Optimal Brain Surgeon and General Network Pruning. In *Proceeding IEEE International Conference on Neural Networks, San Francisco*, pages 293–299, 1993.

[12] Gregory S. Hornby and Jordan B. Pollack. Body-Brain Co-evolution Using L-systems as a Generative Encoding. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 868–875, San Francisco, July 2001. Morgan Kaufmann.

[13] Reinhold Huber, Helmut A. Mayer, and Roland Schwaiger. netGEN - A Parallel System Generating Problem-Adapted Topologies of Artificial Neural Networks by means of Genetic Algorithms. In *Beiträge zum 7. Fachgruppentreffen Maschinelles Lernen der GI-Fachgruppe 1.1.3, Forschungsbericht Nr. 580, Dortmund*, August 1995.

[14] Akio Ishiguro, Siji Tokura, Toshiyuki Kondo, Yoshiki Uchikawa, and Peter Eggenberger. Reduction of the Gap between Simulated and Real Environments in Evolutionary Robotics: A Dynamically–Rearranging Neural Network Approach. In *IEEE Systems, Man, and Cybernetics Conference*, pages III – 239–244. IEEE, October 1999.

[15] Heung Bum Kim, Sung Hoon Jung, Tag Gon Kim, and Kyu Ho Park. Fast Learning Method for Back–Propagation Neural Network by Evolutionary Adaptation of Learning Rates. *Neurocomputing*, 11(1):101–106, 1996.

[16] H. Kitano. Designing neural networks using genetic algorithms with graph generation systems. *Complex Systems*, 4:461–476, 1990.

[17] J. R. Koza. *Genetic Programming: On the Programming of Computers by means of Natural Selection*. Complex Adaptive Systems. The MIT Press, Cambridge, MA, 1992.

[18] Tin-Yau Kwok and Dit-Yan Yeung. Constructive Algorithms for Structure Learning in Feedforward Neural Networks for Regression Problems. *IEEE Transactions on Neural Networks*, 8(3):630–645, 1997.

[19] Helmut A. Mayer. ptGAs–Genetic Algorithms Evolving Noncoding Segments by Means of Promoter/Terminator Sequences. *Evolutionary Computation*, 6(4):361–386, Winter 1998.

[20] Helmut A. Mayer. *Evolutionary Optimization of Components of Artificial Neural Systems*. Habilitation thesis, University of Salzburg, January 2004.

[21] Helmut A. Mayer and Roland Schwaiger. Differentiation of Neuron Types by Evolving Activation Function Templates for Artificial Neural Networks. In *Proceedings of the 2002 World Congress on Computational Intelligence, International Joint Conference on Neural Networks*, pages 1773–1778. IEEE, May 2002.

[22] Geoffrey F. Miller, Peter M. Todd, and Shailesh U. Hegde. Designing neural networks using genetic algorithms. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 379–384, San Mateo, California, 1989. Philips Laboratories, Morgan Kaufman Publishers, Inc.

[23] Marvin L. Minsky and Seymour A. Papert. *Perceptrons: Introduction to Computational Geometry*. MIT Press, Expanded edition, 1988.

[24] David Moriarty and Risto Miikkulainen. Discovering Complex Othello Strategies Through Evolutionary Neural Networks. *Connection Science*, 7(3–4):195–209, 1995.

[25] David E. Moriarty and Risto Miikkulainen. Hierarchical Evolution of Neural Networks. In *Proceedings of the 1998 IEEE Conference on Evolutionary Computation*, pages 428–433, Piscataway, NJ, 1998. IEEE.

[26] Jan Paredis. Steps towards Co–evolutionary Classification Neural Networks. In R. Brooks and P. Maes, editors, *Proceedings Artifical Life IV*, pages 545–552. MIT Press / Bradford Books, 1994.

[27] R. Reed. Pruning Algorithms - A Survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, September 1993.

[28] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the IEEE International Conference on Neural Networks*, San Francisco, CA, April 1993.

[29] L. Rueda and B. John Oommen. The Foundational Theory of Optimal Bayesian Pairwise Linear Classifiers. In *Proceedings of Joint IAPR International Workshops SSPR 2000 and SPR 2000 (LNCS 1876)*, pages 581–590. Springer, 2000.

[30] David E. Rumelhart, Bernard Widrow, and Michael A. Lehr. The Basic Ideas in Neural Networks. *Communications of the ACM*, 37(3):87–92, March 1994.

[31] X. Yao and Y. Liu. A New Evolutionary System for Evolving Artificial Neural Networks. *IEEE Transactions on Neural Networks*, 8(3):694–713, May 1997.

[32] Xin Yao. Evolutionary Artificial Neural Networks. *International Journal of Neural Systems*, 4(3):203–222, 1993.

[33] Xin Yao. Evolving Artificial Neural Networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

[34] Andreas Zell, Guenter Mamier, Michael Vogt, Niels Mach, Ralf Huebner, Kai-Uwe Herrmann, Tobias Soyez, Michael Schmalzl, Tilman Sommer, Artemis Hatzigeogiou, Sven Doering, and Dietmar Posselt. *SNNS Stuttgart Neural Network Simulator, User Manual*. University of Stuttgart, 1994.

[35] Tom Ziemke, Johan Carlsson, and Mikael Bodén. An Experimental Comparison of Weight Evolution in Neural Control Architectures for a 'Garbage-Collecting' Khepera Robot. In *Proceedings of the 1st International Khepera Workshop*. HNI–Verlagsschriftenreihe, 1999.