A Review of Evolutionary Artificial Neural Networks[1][2]

Xin Yao
Commonwealth Scientific and Industrial Research Organisation
Division of Building, Construction and Engineering
PO Box 56, Highett, Victoria 3190
AUSTRALIA

[1]Accepted by **International Journal of Intelligent Systems**, to appear.

[2]Part of this work was done while the author was a Post-Doctoral Fellow at the Computer Sciences Laboratory, Research School of Physical Sciences and Engineering, Australian National University, GPO Box 4, Canberra, ACT 2601, Australia.

Abstract

Research on potential interactions between connectionist learning systems, i.e., artificial neural networks (ANNs), and evolutionary search procedures, like genetic algorithms (GAs), has attracted a lot of attention recently. Evolutionary ANNs (EANNs) can be considered as the combination of ANNs and evolutionary search procedures. This paper first distinguishes among three kinds of evolution in EANNs, 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 on the evolution of connection weights and of architectures, few attempts have been made to understand the evolution of learning rules. Interactions among different evolutions are seldom mentioned in current research. However, the evolution of learning rules and its interactions with other kinds of evolution play a vital role in EANNs. As the final part, this paper briefly describes a general framework for EANNs, which not only includes the aforementioned three kinds of evolution, but also considers interactions among them.

Keywords — Artificial Neural Networks, Learning, Evolution, Genetic Algorithms.

1 Introduction

The interest in EANNs has been growing rapidly in recent years [1, 2, 3, 4], as the research can not only further our understanding of adaptive processes in nature, but also help computer scientists and engineers develop more powerful artificial systems. This paper mainly serves the second purpose. Since we are most interested in exploring possible benefits arising from the interactions between ANNs and evolutionary search procedures, instead of ANNs and evolutionary search procedures themselves, we shall concentrate on the most popular models of ANNs and evolutionary search procedures in our study, i.e., feed-forward ANNs [5] and GAs [6, 7], without trying to cover all kinds of models. However, most discussion is applicable to other models as well, especially when a broader view of evolutionary search procedures is taken, which should include gradient descent based search, heuristic search, and stochastic search like simulated annealing [8, 9], evolution strategies [10], evolutionary programming [11], etc. We shall discuss this issue further in Section 5.

A prominent feature of EANNs is that they can evolve towards the fittest one[1] in a task environment without outside interference, thus eliminate the tedious trial-and-error work of manually finding an optimal (fittest) ANN for the task about which little prior knowledge is available. This advantage of EANNs will become clearer when we discuss the evolution of architectures and of learning rules later. We distinguish among three kinds of evolution in EANNs in this paper, i.e., the evolution of connection weights[2], of architectures and of learning rules, according to the level at which evolutionary search procedures come into EANNs.

Section 2 of this paper is concerned with the evolution of connection weights. The aim here is to find an optimal (near optimal) set of connection weights for an EANN. Various methods of encoding connection weights and their advantages/disadvantages are discussed. Comparison between the evolutionary approach and conventional training algorithms, like back-propagation, is also made.

Section 3 is devoted to the evolution of architectures, which means that an EANN can adaptively find an optimal (near optimal) architecture through an evolutionary process. This kind of evolution provides us with a more powerful adaptive system, which can decide its own architecture according to different tasks to be accomplished. Thus the usual trial-and-error approach used by human designers is replaced by an automatic and systematic one. This work has the same motivation as constructive/destructive learning algorithms [12, 13, 14, 15, 16, 17] have. A review of the

[1] Roughly speaking, the fittest ANN is the one with an optimal (near optimal) architecture, connection weights and learning rule under some optimality criteria.

[2] Thresholds can be considered as connection weights with fixed input -1.

current work is given on the representation of EANN's architectures and genetic operators used to recombine them. Both issues are crucial to the success of the evolution of architectures.

If imagining EANN's connection weights and architectures as its "hardware", it is easier to understand the importance of the evolution of EANN's "software" — learning rules. Section 4 reviews the relationship between learning and evolution, which includes the issue of how learning can guide evolution as well as that of how learning itself can be evolved. It is demonstrated that, started from nearly no ability of learning, an EANN can develop some useful learning rules through evolution.

Although three kinds of evolution mentioned above have been studied independently for several years, few attempts were made to understand the interactions among these three. The EANN as an adaptive system is far from being understood. Section 5 first describes a general framework for EANNs, which includes not only three levels of evolution, but also the interactions among them. Then it concludes with a short summary of this paper.

2 Evolution of EANN's Connection Weights

Learning in ANNs can be divided into supervised and unsupervised. Supervised learning has mostly been formulated as a weight training process, in which effort is made to find an optimal (near optimal) set of connection weights for a network according to some optimality criteria. One of the most popular training algorithms for feed-forward ANNs is back-propagation (BP) [5, 18]. It is a gradient descent search algorithm, which tries to minimise the total mean square error between actual output and target output of an ANN. This error is used to guide BP's search in the weight space. There have been some successful applications of BP algorithms in various areas [19, 20, 21]. However, drawbacks with the BP algorithm do exist due to its gradient descent nature [22]. It often gets trapped in a local minimum of the error function and is very inefficient in searching for a global minimum of a function which is vast, multimodal and nondifferentiable. A detailed review of the current state of the BP algorithm and other learning algorithms can be found in [23].

One way to overcome BP's, as well as other gradient descent search based training algorithms', shortcomings is to consider the training process as an evolution of connection weights towards an optimal (near optimal) set defined by a fitness function and the training task as the environment in which the evolution occurs. From such point of view, global search procedures like GAs can be used effectively to train an EANN. The fitness of an EANN can be defined by the aforementioned total mean

1. Decode each individual (chromosome) in the current generation into a set of connection weights and construct a corresponding EANN with the set (EANN's architecture and learning rule are pre-defined and fixed).

2. Calculate the total mean square error between actual outputs and target outputs for each EANN by feeding training patterns to the EANN, and define $-(error)$ as fitness of the individual from which the EANN is constructed (Other fitness definitions can also be used, depending on what kind of EANNs is needed).

3. Reproduce a number of children for each individual in the current generation with probability according to its fitness or rank, i.e., using the roulette wheel parent selection algorithm [7] or Whitley's rank based selection algorithm [34].

4. Apply genetic operators, such as crossover, mutation and/or inversion, with probability to child individuals produced above, and obtain the new generation.

Figure 1: A typical cycle of the evolution of connection weights.

square error. The selective pressure in such evolution is against those EANNs which are less fit (have large errors). The main idea here is to use GAs as function optimisers to maximise fitness functions (minimise error functions), since GAs are good at dealing with large, complex, nondifferentiable and deceptive spaces. A lot of work has been done along this line [24, 25, 26, 27, 28, 29, 30, 31, 32, 33].

The evolutionary training approach is divided into two major steps; the first one is to decide the representation scheme of connection weights, e.g., whether it is in the form of binary strings or not, and the second one is the evolution itself driven by GAs. Different representation schemes and GAs can lead to quite different training performance in terms of training time and accuracy. A typical cycle of the evolution of connection weights is shown in Figure 1.

2.1 Representation of Connection Weights as Binary Strings

The most convenient representation of connection weights is, from GA's perspective, binary strings [24, 26, 29]. In such a representation scheme, each connection weight is represented by some binary bits with certain length. For example, Whitley *et al.* used 8 bits to represent each connection weight, which ranges between -127 and $+127$, in their experiments with XOR and adder problems [24, 29]. The set of connection weights in an EANN is simply represented by the concatenation of all the weights in the network in binary form. The order of the concatenation is, however, essentially ignored, although it can affect the performance of evolutionary training, e.g., training time and accuracy [31].

After the representation scheme has been decided, a GA is used to evolve a population of individuals (sets of connection weights) as shown in Figure 1, but EANN's architecture, which includes the number of hidden nodes, node transfer functions, and the connection topology among all nodes, and its learning rule are pre-defined and fixed during the evolution. Each individual's fitness is evaluated by the EANN constructed from it. According to GA's lingoes [7], it is the *genotype* (encoded connection weights), instead of the *phenotype* (EANNs constructed), whose fitness is needed in the evolution. Fortunately, the genotype is equivalent to the phenotype in this case because EANN's architecture and learning rule are fixed and the same for all EANNs. We shall return to this point in Section 3 when we discuss the evolution of EANN's architectures, where the evaluation of the genotype is not equivalent to, but approximated by, the evaluation of the phenotype.

The evolutionary training process described in Figure 1 adopts so-called *batch* training mode, i.e., weights are changed only after all training patterns have been presented to the EANN. This is different from most sequential training algorithms, like sequential BP, where weights are updated after each training pattern is presented to the network. Batch training mode is particularly suitable for parallelisation of training process.

The binary encoding of connection weights need not be uniform as adopted by Whitley *et al.* it can also be Gray, exponential or more sophisticated [31]. A limitation of binary representation is the representation precision of discretised connection weights. If too few bits are used to represent each connection weight, training may take an extremely long time, or even fail, because some combinations of real connection weight cannot be approximated by discrete values within certain tolerant range. On the other hand, if too many bits are used, binary strings representing large EANNs become very long, which will prolong the evolution dramatically and make the evolutionary training approach impractical. It is still an open question how to optimise the number of bits for each connection weight, the range encoded, and the encoding method used although dynamic encoding techniques [35] could be adopted to alleviate the problem. This problem is closely related to *weight quantisation*, which has been studies using BP [36], Boltzmann machines [37], and the Cascade-Correlation algorithm [38], where the major purpose is to facilitate the digital VLSI/optical implementation of ANNs. A result of general interest will be the comparison of impact of weight quantisation on training and generalisation among different ANN models and training algorithms (including GAs).

It is well known that nodes in a hidden layer of a multilayer feedforward ANN function basically as input feature detectors. Separation of encoded connection weights leading to the same hidden node far apart in the binary representation will make the exploitation of nonlinear interactions among them much more difficult because of the interruption of crossover [31]. Hence, much more time will be needed to evolve useful feature detectors (connection weights). It has been demonstrated that concatenation of encoded connection weights according to hidden nodes is beneficial in exploiting and keeping useful *functional blocks* formed around hidden nodes [39]. Various kinds of adaptive crossover [40, 41, 42], which have different probabilities at different points in a string, can further alleviate the functional block problem, i.e., the problem of crossover's disruption and destroy of functional blocks. Crossover probability can be made higher at boundary points between two functional blocks, instead of uniform at every point in the string.

Although there are problems with the binary representation of connection weights, as indicated before, some successful experiments with small EANNs have been carried out [24, 29, 31]. These experiments also showed that some ANNs trained successfully by the evolutionary approach cannot be trained within a tolerable time by conventional BP algorithms.

2.2 Representation of Connection Weights as Real Numbers

To overcome some shortcomings of the binary representation scheme, real numbers themselves were proposed to represent connection weights, i.e., one real number per connection weight [25, 27, 28]. An EANN is represented by the concatenation of these real numbers, where the order of concatenation is important as explained in the last section. Various kinds of adaptive crossover mentioned in the last section are also applicable here, but they are unlikely to change a single connection weight because they are seldom applied, in practice, at a point within a real number, although crossover can theoretically break at any point between digits regardless of real or binary encoding representation. Single real numbers are often changed by average crossover, random mutations, real number creep and/or other domain-specific genetic operators [43].

As shown above, standard genetic 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 EANN's training, in order to improve the speed and accuracy of the evolutionary training. Montana and Davis defined a large number of domain-specific genetic operators, which incorporated many heuristics about training ANNs, in their study [25]. The major aim was to retain useful functional blocks during evolution, i.e., to form and keep useful feature detectors in an EANN. Their results showed that the evolutionary training approach was much faster than BP training algorithms, at least for the problem they considered, although the domain-specific genetic operators they used might not have very good generality. The results also illustrated how domain knowledge could be introduced into evolutionary search procedures to improve their performance.

Similar results were obtained by Bartlett and Downs, which showed that the evolutionary approach was faster, and the larger an EANN, the greater the speedup over BP algorithms [28]. This implies that the scalability of evolutionary training is better than that of BP training, but more work need to be done to confirm such a claim in the general case.

The real representation scheme does not mean that a complex set of genetic operators is indispensable. Simple sets of genetic operators can equally be used in evolutionary training. For example, Fogel *et al.* adopted only one genetic operator (excluding selection) — Gaussian random mutation, a widely used creep operator which adds a Gaussian random number within certain range to the original weight — in their evolutionary training of EANNs [27], which was based on the original evolutionary programming concept [11]. Since there is no crossover in evolutionary programming, it is essentially equivalent to the implementation of multiple (i.e., a population of) simulated annealing in parallel. The Cauchy random mutation might be a better choice than the Gaussian one here because it offers faster convergence [44, 9].

2.3 Comparisons Between Evolutionary Training and BP Training

As indicated at the beginning of Section 2, the evolutionary training approach is attractive because it can handle the global search problem better in a vast, complex, multimodal and nondifferentiable (connection weight) space. It is usually based on a global search algorithm, such as GA, thus can escape from a local minimum. It does not need to calculate derivatives of the error function, thus works well with nondifferentiable error functions. It sets no restriction on types of ANNs being trained as long as a suitable fitness function can be defined properly, thus can deal with a wide range of ANNs.

An often used method to decrease ANN's complexity and improve its generalisation ability is to add a penalty term in the error function. This can be achieved in a straightforward way in the evolutionary training approach by just adding the penalty term to the fitness function without worrying about the calculation of its derivatives to each connection weight. This simple method is applicable to a wide range of ANNs, not just feedforward ones. Other requirements can also be incorporated into the fitness function in a similar way to train EANNs with some special characteristics.

There is, however, computational cost for the above advantages. Evolutionary training is usually computation more intensive and slower than gradient descent based training, such as BP. Kitano carried out a series of empirical comparisons between using GAs and using fast variants of the back-propagation algorithm to train EANNs [45]. He showed that GAs normally took longer time to converge than fast back-propagation algorithms did, thus were less efficient. This is not surprising since evolution is an adaptive process which is more suitable for working on the slow time scale than on the fast time scale. That is, it is more suitable for modelling adaptation to changes of an environment rather than to the environment itself. From the algorithmic point of view, GAs are good at global sampling, instead of local fine-tuning. Hence, GAs are often outperformed by some fast gradient descent algorithms, with regard to the speed of convergence in training, although they might enjoy global convergence theoretically.

It should be kept in mind that the above comparisons are based on experiments with three-layer feedforward ANNs. It is indicated that the evolutionary training approach is more efficient than BP training for deep feedforward ANNs (ANNs with more than one hidden layers) and non-feedforward ANNs [29, 32]. Because no search procedure is the overall winner in attacking all sorts of error (fitness) functions, and each search procedure is only suitable 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. An immediate question concerning GAs is: What kind of error (fitness) function is GA-hard and what is GA-easy? The answer to this question and the like will help us better understand the merits of different training approaches.

2.4 Hybrid Evolutionary Training Approach

The efficiency of evolutionary training can be improved by incorporating a local search procedure into the evolution, i.e., combining GA's global sampling ability with local search's fine-tuning ability. GAs can first be used to quickly locate a good region, represented by a starting point, in the weight space, and then local search is employed to find a near optimal solution in this region. The local search procedure can be a gradient descent search algorithm like BP [31, 45] or a more powerful search algorithm like simulated annealing [46].

Belew *et al.* used GAs to search for a good set of initial connection weights and then used BP algorithms to do further fine-tuning from these initial values [31]. Their results showed that the hybrid GA/BP approach was more efficient than GAs and also very competitive in comparison with BP algorithms. Although the hybrid evolutionary approach still seems to need more computation time than that BP algorithms need, it is in fact not the case because BP algorithms used to run multiple times to get a good solution due to BP's sensitivity to the initial weights. GAs are much better at locating good initial weights than the random start method.

The comparison Kitano did between the hybrid GA/BP approach and the BP algorithm is somewhat unfair [45], because he used standard BP [5] as the local search procedure in the hybrid GA/BP training while adopted a fast variant of BP — Quickprop [47] for training independently, although the conclusion that Quickprop converges faster than the hybrid approach does might still be true if the same local search procedure, Quickprop, is adopted in the hybrid approach. A key issue addressed by Kitano was whether GAs converge faster than Quickprop in the initial stage of weight space search. If the answer is no, there will be no advantage, in the speed of convergence, of using GAs to locate good initial connection weights. This is, unfortunately, the conclusion drawn from Kitano's experiments.

An important reason of the slow convergence of GAs is the lack of compositional feature, the feature that good partial solutions (schemata) can be recombined into better overall solutions, in the evolution of connection weights, because of interdependencies among sections of chromosomes. This is closely related to the functional block problem mentioned in Section 2.1. A locally good partial solution which is smaller than a functional block is not necessarily good when evaluated from the point of view of a whole EANN after recombination (especially crossover), since the close interactions within a functional block are interrupted and destroyed. In contrast, a good functional block, an actual feature extractor in EANNs, is much more likely, although not always, to be part of a fit EANN.

Despite of the negative conclusion from Kitano's experiments with some artificial problems that the hybrid GA/BP approach is still slower than Quickprop, it is unclear whether the conclusion is also true for real-world applications. Moreover, GA/BP hybridisation can take place at any point in a whole spectrum, where one end is pure GA and the other end is pure BP. The optimal hybridisation of GA's exploration ability and BP's exploitation ability is an example of a more general research issue of exploration versus exploitation in search and is highly problem dependent. Some preliminary experiments have suggested that a hybridisation point close to the pure local search end be used in training [46].

3 Evolution of EANN's Architectures

Section 2 assumes that EANN's architecture is pre-defined and fixed during the evolution of connection weights. But how to decide EANN's architecture? It is well known that EANN's architecture has significant impact on EANN's information processing abilities. Unfortunately, EANN's architecture still has to be designed by experienced experts through trial-and-error. There is no systematic way to design an optimal (near optimal) architecture for a particular task. Recent research on constructive/destructive algorithms is one of the many efforts made towards the automatic design of EANN's architectures [12, 13, 14, 15, 16, 17]. Roughly speaking, a constructive algorithm starts with a minimal network (network with minimal number of hidden units and connections) and adds new nodes and connections if necessary during training, while a destructive algorithm does the opposite, i.e., starts with the maximal network and delete unnecessary nodes and connections during training.

The optimal design of EANN's architecture can be viewed as searching for an architecture which performs best on a specified task according to some optimality criteria, that is, searching the surface defined by the optimality level of EANN's architecture in the architecture space, which is composed of all possible architectures. There are several characteristics with such a surface, as indicated by Miller *et al.* [48], which make the GA-based evolutionary approach a better candidate for searching the surface than the heuristic approach, like aforementioned constructive/destructive algorithms. These characteristics include [48]

- The surface is *infinite large* since the number of possible nodes and connections is unbounded.
- The surface is *nondifferentiable* since changes in the number of nodes or connections is discrete and can have a discontinuous effect on EANN's performance (optimality).
- The surface is *complex* and *noisy* since the mapping from EANN's architecture to EANN's performance after training is indirect, strongly epistatic, and dependent on initial conditions.
- The surface is *deceptive* since EANNs with similar architectures may have dramatically different information processing abilities and performances.
- The surface is *multimodal* since EANNs with quite different architectures can have very similar capabilities.

Because of advantages of the evolutionary design of architectures, a lot of research has been carried out in recent years [49, 50, 48, 51, 52, 53, 54, 55, 56, 57, 58, 59, 29, 60, 32, 61, 62, 63], which concentrates on the evolution of EANN's connectivity, i.e., the number of nodes in an EANN and the connection topology among these nodes. Little work, however, has been done on the evolution of node transfer functions except for a couple of non-GA-based approaches to it [64, 65], let alone the evolution of both connectivity and node transfer function. We shall only cover the evolution of node transfer functions briefly in Section 3.3 since the emphasis of this paper is on GA-based approaches to evolution. Node transfer functions are pre-defined and unchanged during the evolution of connectivity in Section 3.1 & 3.2 unless specified explicitly.

Similar to the evolutionary training approach, the first step of the evolutionary design of architectures is to decide a proper representation of architectures. But the problem now is not whether to use a binary representation or a real one since we only deal with discrete values as mentioned before and binary representation is required. The problem is more related to the conceptual structure of representation, e.g., a matrix, a graph, or some generation rules. A key issue here is to decide how much information about an architecture should be encoded into a representation. At one end, all the information about an architecture is represented directly by binary strings, i.e., each connection and node is specified directly by some binary bits. This kind of representation is called *direct encoding scheme*[3]. At the other end, only the most important parameters or features of an architecture are represented, such as the

[3]There are different names for such representation, e.g., Miller *et al.* call it *strong specification scheme* and call the *indirect encoding scheme* the *weak specification scheme* [48].

1. Decode each individual in the current generation into an architecture with necessary details, in the case of the indirect encoding scheme; supplied by either some *developmental rules* or the training process.

2. Train each EANN with the decoded architecture by a pre-defined and fixed learning rule (but some parameters of the learning rule may be adaptive and learned during training), starting from different sets of random initial values of connection weights and, if any, learning rule parameters.

3. Calculate the fitness of each individual (encoded architecture) based on the above training results; e.g., based on the smallest total mean square error of training, or testing if more emphasis is laid on generalisation, the shortest training time, the architecture complexity (fewest nodes and connections and the like), etc.

4. Reproduce a number of children for each individual in the current generation with probability according to its fitness or rank.

5. Apply genetic operators, such as crossover, mutation and/or inversion, with probability to child individuals produced above, and obtain the new generation.

Figure 2: A typical cycle of the evolution of architectures.

number of nodes, the number of connections, and the type of node transfer function. Other details of the architecture are left to the learning (training) process to decide. Such kind of representation is called *indirect encoding scheme*. The evolution of architectures can be described by the cycle shown in Figure 2.

3.1 Direct Encoding Scheme for EANN's Connectivity

In direct encoding scheme, each connection of an EANN is specified directly by its binary representation [52, 51, 48, 59, 60, 29, 62, 63]. Because the chromosomal representation of EANN's connectivity has specified all the detailed information about it, the developmental rule used to decode chromosomes into EANN's connectivity patterns is virtually degenerated into a one-to-one mapping, no real "development" at all. In general, an $N \times N$ matrix $C = (c_{ij})_{N\times N}$ can represent the connectivity of an EANN with N nodes, where c_{ij} indicates presence or absence of the connection from node i to node j. We can use $c_{ij} = 1$ to stand for a connection and $c_{ij} = 0$ for no connection. Each such matrix has a direct one-to-one mapping to the corresponding connectivity. The binary string representing EANN's connectivity is just the concatenation of rows (or columns) of the matrix. Some restraints on EANNs being explored can be incorporated easily into such representation scheme, e.g., entry c_{ij} of the matrix can specify that only a positive/negative weight is allowed if a connection is present [48]. The matrix can also be restrained in such a way that only feedforward connections are allowed.

The direct encoding scheme is suitable for the precise and deterministic handling of small EANNs, i.e., those with small number of nodes. It may facilitate the rapid generation and optimisation of tightly pruned, interesting designs that no one has hit upon before [48]. It, however, does not scale well since large EANNs need very large matrices to represent, which make the evolution of connectivity much slower. A natural way to cut down the size of matrices is to use as much domain knowledge as possible to restrain the search space. For example, if the connection pattern, like the complete one, between input (output) and the hidden layer is known for three-layer feed-forward EANNs, then only information about the number of hidden nodes needs encoding, thus the length of the corresponding binary string can be reduced greatly[51, 52]. However, this kind of reduction is very limited and only effective when there is enough prior knowledge about EANN's connectivity, which is often not the case in practice.

The functional block problem mentioned in Section 2.1 still exists in the evolution of connectivity because blocks formed around hidden nodes are quite vulnerable to crossover, unless some techniques like adaptive crossover are used to lessen such vulnerability. One method to measure the severeness of the functional block problem is to look at the probability of generating unfit children from fit parents by crossover. This is also a measurement of the compositionality, first introduced in Section 2.4, of encoded connectivity patterns.

As indicated in Section 2.3, an advantage of the evolutionary approach is that the fitness function can be defined easily in such a way that an EANN with some special features is evolved. For example, EANNs with better generalisation can be obtained if testing results, instead of training results, are used in their fitness calculations [51]. A penalty term in the fitness function for complex connectivity can also help improve EANN's generalisation ability, besides the cost benefit, by reducing the number of nodes and connections in EANNs. Schaffer *et al.*'s experiments showed that EANNs designed by the evolutionary approach have better generalisation ability than EANNs trained alone by BP algorithms with pre-defined connectivity [51], but more studies are needed to support such a conclusion since there also exist methods to improve BP network's generalisation ability.

Fitness functions based on information theory have been studied recently, although a non-evolutionary approach was adopted in the selection of EANN's architectures [66, 67, 68]. Due to the importance of *parsimony* in achieving better generalisation of a learning system [69], fitness functions defined by various information criteria seem to offer promising EANNs with better generalisation ability. We are currently investigating the evolution of both architectures and connection weights based on the fitness function defined by the *Minimum Description Length* principle [70]. Other criteria used include *AIC* (Akaike's Information Criterion) [67, 68], and *conditional class entropy* [66], but more efficient algorithms need to be studies in order to search for the fittest architecture in the fitness space defined by these criteria since no automatic search algorithm, except for a manual selection method, was given in [67, 68]. Simulated annealing was adopted by Bichsel and Seitz to search for both the number of hidden nodes and connection weights in feed-forward EANNs [66].

It should be noted that information criteria are equally applicable to the indirect encoding scheme discussed in the next section although they first appear here.

3.2 Indirect Encoding Scheme for EANN's Connectivity

A good way to reduce the length of connectivity representation is to encode only most important features, instead of each individual connection, of an connectivity pattern [50, 54, 57, 53, 58, 49, 71, 72]. Details about each connection are supplied by some developmental rules during chromosome decoding. An immediate benefit of the indirect encoding scheme is a compact representation of EANN's connectivity. The scheme is also biologically more plausible than the direct encoding one because it is impossible for genetic information encoded in chromosomes to specify the whole nervous system directly and independently according to the discoveries of neuroscience [71]. For example, the human genome has estimately $30,000$ genes with an average of $2,000$ base pairs each [73], that is roughly 10^8 in total; this is clearly insufficient to specify about 10^{15} synapses directly and independently in the human brain [74].

3.2.1 Encode Connectivity Parameters

There are various sets of parameters which can be used to specify connectivity. Harp *et al.* adopted *blueprints*, a kind of binary indirect representation which is composed of one or more segments, to encode EANN's connectivity and learning parameters used by the BP training algorithm [50, 54, 75]. Each segment consists of two parts: (1) an area parameter specification (APS) which is of fixed length and parameterises the area (i.e., layer in a feed-forward EANN) in terms of its address, the number of nodes in it, the organisation of these nodes, and learning parameters associated with the nodes; and (2) one or more projection specification fields (PSFs) which describe the efferent connectivity (projections) from the current area to one or more other areas in terms of connection density, the target area address, the organisation of connections, and learning parameters associated with these connection weights. The first and last areas are constrained to be the input and output areas respectively. The length of a blueprint is variable because the number of areas is not pre-defined. Area and projection markers are used to segment different areas and projections.

It can be seen from the above that only parameters of an connectivity pattern, instead of each individual connection, is specified by a blueprint. The detailed node-to-node connection is specified by certain implicit developmental rules, e.g., the network instantiation software used by Harp *et al.*. Similar parametric representation of connectivity has also been studied by Hancock [57] and Dodd *et al.* [53], but different parameterisation methods were used. An interesting aspect of Harp *et al.*'s encoding scheme is their combination of learning parameters into the connectivity representation, which, in fact, explores the interaction between the evolution of connectivity and the evolution of learning rules. We shall discuss this issue further in Section 4.

Although the aforementioned indirect representation methods can reduce the length of binary strings specifying EANN's connectivity, they still have not got the necessary scalability needed by many real-world applications since the length grows quickly with large EANNs. The issue of optimal parameterisation of connectivity and that of representing each parameter with the minimum number of bits while not greatly restricting exploration of possible useful connectivity patterns are still open for further research. Moreover, developmental rules play less important role here because they are in essence some fixed assumptions made based on our prior knowledge about connectivity we would like to obtain.

3.2.2 Encode Developmental Rules

A quite different indirect encoding method from that described in Section 3.2.1 is to encode developmental rules in the connectivity representation [49, 71], where the major aim is to optimise developmental rules which can lead to the construction of optimal connectivity patterns, instead of to optimise connectivity patterns themselves. The shift from the direct optimisation of connectivity patterns to the optimisation of developmental rules for constructing connectivity patterns can bring out many advantages, such as better scalability of the method and better regularity and generalisation ability of the resultant EANN, since rules do not grow with the size of EANNs. The functional block problem caused by crossover will also be less severe because the rule encoding method concerns much more about connections among groups of nodes than about those among single nodes, and is capable of preserving promising functional blocks found so far [49].

A developmental rule in the rule encoding method is usually described by a recursive equation [71] or a generation rule similar to a production rule in a knowledge based system with a left-hand side and a right-hand side [49]. EANN's connectivity pattern, in the natural form of matrix, is constructed from a basis, i.e., a single element matrix, by repetitively applying suitable developmental rules to non-terminal elements in the current matrix until the matrix contains only terminal[4] elements which indicate the presence or absence of a connection, that is, a connectivity pattern is fully specified.

Kitano [49] used a modified version of graph generation system [76], which includes a set of graph generation rules, to construct connection matrices since each connection matrix corresponds to a directed graph. Each developmental rule he used, i.e., a graph generation rule, consists of a left-hand side (LHS), which is a non-terminal element, and a right-hand side (RHS), which is a 2×2 matrix with either terminal or non-terminal elements. A typical step of constructing a connection matrix is to find the rules whose left-hand sides appear in the current matrix and replace all the appearance with respective right-hand sides. Each rule is represented by five *allele* positions, corresponding to five elements in a rule, in the chromosome. The length of chromosome is variable with the first position fixed to an initial element, i.e., the basis. Each position in the chromosome can take the value of an element in the range from "A" to "p". The 16 rules with "a" to "p" on the left-hand side and 2×2 matrices with only 1's and 0's on the right-hand side are pre-defined and do not participate in evolution. That is, not all developmental rules are obtained through evolution.

Consistently better results with various size encoder/decoder problems were reported by Kitano in comparison with the direct encoding scheme. However, the rule

[4] In this paper, a terminal element is either 1 (existence of a connection) or 0 (non-existence of a connection), and a non-terminal element is a symbol other than 1 and 0. These definitions are slightly different from those used by Kitano [49].

encoding method, as expected, is not very good at fine-tuning detailed connections among single nodes because it concentrates on connections among groups of nodes. The addition of a fine-tuning process, in which a dynamic node readjustment algorithm similar to constructive or destructive algorithms [12, 13, 14, 15, 16, 17] could be used, after the evolution seems to be a promising approach to better combination of global structure evolution and local fine-tuning.

Mjolsness *et al.* described a similar rule encoding method where rules are represented by recursive equations which specify the growth of connection matrices [71]. Coefficients of these recursive equations, represented by decomposition matrices, are encoded in chromosomes and optimised by simulated annealing, instead of GAs. As a matter of fact, connection weights are optimised along with connectivity by simulated annealing because each entry of a connection matrix can have a real weight, rather than just 1 or 0.

The approach of optimising connectivity and connection weights at the same time has also been adopted by Bornholdt and Graudenz although they did not use the rule encoding method for connectivity [62]. In each cycle of their evolutionary approach, a number of nodes or connections (with weights) are added to or deleted from the current EANN at random, where the EANN is specified by both connectivity and connection weights encoded in the chromosome. The only genetic operator mentioned in their paper is mutation, i.e., the above random addition or deletion. After mutation, the EANN is evaluated directly by feeding with all training examples without additional training by BP or other algorithms since the chromosome has already specified connection weights. Because only some toy problems were tried by Bornholdt and Graudenz with a long computation time, it is hard to judge the merits of this approach without further investigation.

3.2.3 Fractal Representation of Connectivity

Merrill and Port proposed another method for encoding connectivity, which is based on the use of fractal subsets of the plane [72]. They argued that the fractal representation of connectivity was biologically more plausible than the rule encoding representation because there is evidence that part of the human body, like lung [77], is fractally structured. It is unlikely, however, that their method will have better scalability than the rule encoding method since they used three real parameters, i.e., an edge code, an input coefficient and an output coefficient, to specify each node in a connectivity pattern. Fast simulated annealing [44] was adopted to optimise the fractal representation.

3.3 Evolution of Node Transfer Functions

As indicated in Section 3, almost all the work done on the evolution of EANN's architectures is actually about the evolution of EANN's connectivity. Little work has been done on the evolution of node transfer functions, although it has been shown that the transfer function is an important part of an architecture and has significant impact on EANN's performance [64, 65]. The node transfer function normally stays the same during the evolution of connectivity.

Mani proposed a modified BP which performs gradient descent search in the weight space as well as the transfer function space [64], but EANN's connectivity is fixed. Lovell and Tsoi investigated the performance of Neocognitrons with various S-cell and C-cell transfer functions, but did not adopt any adaptive procedure to search for an optimal transfer function automatically [65]. Stork *et al.* are, to our best knowledge, the first to apply GA-based approach to the evolution of both connectivity and node transfer functions even though only very simple neural networks consisting of 7 nodes were investigated [61].

In principle, transfer functions of different nodes in an EANN can be different and decided automatically by an evolutionary process, instead of assigned by human experts. The difference in transfer functions could be as large as that in the function type, e.g., that between a hard-limiting threshold function and a Gaussian function, or as small as that in one of parameters of the same type of function, e.g., the slope parameter of the sigmoid function. The decision on how to encode transfer functions in chromosomes depends on how much prior knowledge and computation time are available. In general, nodes within a group, like a layer, in an EANN tend to have the same type of transfer function with possible difference in some parameters, while different groups of nodes might have different types of transfer function. 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.

One point worths mentioning here is the evolution of both connectivity and transfer functions at the same time [61] since they constitute a complete architecture. Encoding connectivity and transfer functions into the same chromosome makes it easier to explore nonlinear relations between them. Many techniques used in encoding and evolving connectivity could equally be used here.

3.4 Discussions

The representation of EANN's 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. As indicated before, crossover can destroy useful functional blocks during the evolutionary process. It can also lead to the generation of unfeasible solutions, e.g., architectures with no connection path from input to output, if applied blindly. The indirect encoding scheme and adaptive crossover lessen such damage to some extent, but cannot avoid it completely. An alternative is to employ algorithms without crossover, like simulated annealing, in the evolutionary process.

Another problem associated with the representation issue is the so-called "hidden node problem", i.e., hidden node labels are arbitrary [32, 31]. There is no easy way to recognise the situation that two different binary strings represent, in fact, the same architecture because they only differ in the order of labelling hidden nodes. Such invariance under permutation of hidden nodes causes a severe problem that exhibits enormous redundancy in the architecture space. Unfortunately, no satisfying technique has been implemented to tackle this problem.

Unlike the fitness evaluation of encoded connection weights, the fitness evaluation of encoded architectures is very noisy because what has actually been evaluated is phenotype's fitness, i.e., the fitness of individual EANNs with their architectures decided by chromosomes and developmental rules and their initial connection weights generated at random, which is only a rough approximation of genotype's fitness, i.e., the fitness of encoded architecture without any stochastic component, due to the nondeterministic nature of random initial connection weights [53]. In other words, we want to optimise the genotype so that it can perform well regardless of initial connection weights, but we can only approximate such optimisation by examining phenotypes with limited sets of initial connection weights out of a virtually indefinite number of sets. This problem can be circumvented by either encoding initial connection weights as part of an architecture or combining the evolution of connection weights and that of architectures into one, i.e., encoding and evolving connection weights and architectures together without employment of another weight training algorithm [71, 62].

It has been widely accepted that 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. Research in neuroscience has indicated that recombination like crossover is best performed between groups of neurons rather than individual neurons [78]. But it is still an open question that how large a group should be. The answer to this question has significant impact on the level of chromosomal representation. In general, the larger the group, the higher the level and the more indirect the encoding. Thus, more powerful developmental rules are needed to specify the internal structure of a large group. A further question here is how to group neurons adaptively during evolution instead of guess and fix the group size before evolution.

4 Evolution of EANN's Learning Rules

It is known that different architectures and learning tasks[5] need different training algorithms, e.g., GAs are suitable for training EANNs with feedback connections and deep feedforward EANNs (EANNs with many hidden layers), while BP is good at training shallow ones. Even after selecting a training algorithm, there are still algorithm parameters, like the learning rate and momentum in BP algorithms, which have to be specified. The optimisation of training algorithms and their parameters for an EANN and a learning task is usually very hard because little prior knowledge about EANN's architecture and the learning task at hand is available in practice.

Some work has been done on adaptively adjusting BP algorithm's parameters, such as the learning rate and momentum, through the heuristic or evolutionary approach [79, 75, 31], but the more fundamental issue of optimising the learning rule, i.e., the weight-updating rule as sometimes called, which underlies the learning algorithm has only been addressed by a limited number of researchers [80, 81, 82, 83]. Even though the Hebbian learning rule [84] is widely accepted and used as the basis of many learning algorithms, recent research by Hancock *et al.* [85] shows that another learning rule based on Artola *et al.*'s work [86] is more powerful than the optimal Hebbian rule. It can learn more patterns than the optimal Hebbian rule and can learn exceptions as well as regularities. 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. It is very appealing to develop an automatic method of optimising learning rules for an EANN and an learning task.

The evolution of mankind's learning ability from relatively weak to very powerful suggests the potential benefit of introducing an evolutionary process into EANN's learning. The relationship between evolution and learning in such a combined system is extremely complex and has been investigated by many researchers [87, 88, 89, 90, 91, 92, 93, 94, 80, 81, 82, 83, 95]. Most research, however, concentrates on the question

[5] Learning tasks (algorithms) have the same meaning as training tasks (algorithms) in this paper.

1. Decode each individual in the current generation into a learning rule which will be used to train EANNs.
2. Construct a set of EANNs with randomly generated architectures and initial connection weights, and evaluate them by training with the decoded learning rule, in terms of training or testing accuracy, training time, architecture complexity, etc.
3. Calculate the fitness of each individual (encoded learning rule) based on the above evaluation of each EANN, e.g., by some kind of weighted averaging.
4. Reproduce a number of children for each individual in the current generation with probability according to its fitness or rank.
5. Apply genetic operators, such as crossover, mutation and/or inversion, with probability to child individuals produced above, and obtain the new generation.

Figure 3: A typical cycle of the evolution of learning rules.

of how learning can guide evolution [87, 88, 89, 90, 91] or the relationship between the evolution of architectures and weight training [92, 93, 94], rather than the evolution of learning rules, which has only attracted limited attention [80, 81, 82, 83]. Apart from offering an approach of optimising learning rules, the evolution of learning rules is also important in modelling the relationship between learning and evolution and modelling the creative process, since newly evolved learning rules have the potentiality to deal with a complex and changing environment. The research on the evolution of learning rules will help us to better understand how creativity can emerge from artificial systems like EANNs and how to model the creative process in biological systems. A typical cycle of the evolution of learning rules can be described by Figure 3.

Similar to the case in the evolution of architectures, the fitness evaluation of each learning rule is also very noisy because randomness is introduced into the evaluation by not only initial connection weights but also architectures. Even if a particular architecture is pre-defined and fixed during evolution, like most people have assumed [80, 81, 82, 83], noise still exists due to random initial connection weights.

4.1 Evolution of BP Algorithm's Parameters

The adaptive adjustment of BP algorithm's parameters, like the learning rate and momentum, through evolution could be considered as the first attempt of the evolution of learning rules [50, 31]. Harp *et al.* encoded BP algorithm's parameters in chromosomes together with EANN's architectures [50]. An effect of such an encoding strategy is the further exploration of interactions between learning algorithms and architectures, so that an optimal (near optimal) combination of a BP algorithm with an architecture can be evolved. Belew *et al.* also used an evolutionary process to find parameters of the BP algorithm, but EANN's architecture is pre-defined [31]. The parameters evolved in this case tend to be optimised towards the architecture, instead of general applicable ones because part of the environmental diversity is lost by fixing the architecture.

4.2 Evolution of Learning Rules

Whenever a kind of evolution is introduced into EANNs, a chromosomal representation scheme has to be developed. It is much more difficult to encode dynamic behaviours, like the learning rule, than to encode static properties, like the architecture and connection weights, of an EANN. Trying to develop a universal representation scheme which can specify any kind of dynamic behaviours of an EANN is clearly impractical, let alone the prohibitive long computation time required to search such a learning rule space. Constraints have to be set on the type of dynamic behaviours, i.e., the basic form of learning rules, being evolved to limit the representation complexity and the search space.

Two basic assumptions which have often been made on the learning rule are: (1) The weight-updating of a connection depends only on local information, such as the activation of the input node, the activation of the output node, the current connection weight, etc.; and (2) The learning rules is the same for all connections in an EANN. 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

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

where t is time, Δw is the weight change, $x_1, x_2, \cdots, x_n$ are local variables, and θ's are real coefficients. The major aim of the evolution of learning rules is to decide these coefficients.

Due to the large number of terms in (1), which can make the evolution extremely slow and impractical, further constraints are often set based on either biological or other heuristic knowledge [80, 81, 82, 83]. Chalmers defined the form of learning rules as a linear function of four local variables and their six pairwise products [80]. No third or fourth order[6] term was used. Ten coefficients and a scale parameter are encoded in a binary string via exponential encoding. The architecture used in learning rule evaluation is fixed because only single layer EANNs are considered and the number of inputs and outputs are fixed by the learning task at hand, although the architecture could be generated at random and even evolved at the same time when the learning rule is evolved. After 1000 generations, starting from a population of randomly generated learning rules, represented by 11 coefficients, the evolutionary process discovers the well-known delta rule [96] and some of its variants. These experiments, although simple and preliminary, have demonstrated the potentiality of the evolution of learning rules to discover novel learning rules, not merely known ones. However, the constraints set on the form of learning rules could prevent some learning rules from being evolved, like those included third or fourth order terms.

Similar experiments on the evolution of learning rules were also carried out by Fontanari and Meir [83] and Bengio *et al.* [81, 82]. Fontanari and Meir used Chalmers' approach to evolve a learning rule for the binary perceptron. They also considered four local variables, but only 7 terms were adopted in their weight-updating function, which include one first order, three second order and three third order terms in (1). Bengio *et al.*'s approach is slightly different from Chalmers' in the sense of that gradient descent algorithms and simulated annealing, instead of GAs, were employed as a means of evolution. Four local variables were considered as usual. One zeroth order, three first order and three second order terms in (1) were used in their weight-updating function.

It should be noted that the environment in which the learning rule evolves includes both the learning task and the architecture. If a general optimal (near optimal) learning rule which is applicable to a wide range of architectures and learning tasks needs to be evolved, the environmental diversity has to be high enough. That is, enough different, representative architectures and learning tasks have to be presented when evaluating the fitness of a learning rule. Unfortunately, it is still an open question that how many can be considered as "enough". The role of environmental diversity has also been studied by Parisi *et al.* [90, 97]. Their work on *econets* could be viewed as another effort in understanding the evolution of learning although they did not represent and evolve learning rules explicitly.

[6]The order here is defined as the number of variables in a product.

4.3 Evolution of Evaluation Functions

Supervised learning has been assumed in our above discussion of EANNs, but reinforcement learning and unsupervised learning can also be combined with evolution to generate more effective learning algorithms. Unlike supervised learning where full feedback about desired actions, which enables more accurate evaluation of a learning algorithm or system, is provided by external teachers, reinforcement learning only has a scalar feedback representing the utility of a given action, which makes accurate evaluation quite challenging.

Ackley and Littman proposed a novel learning strategy, called evolutionary reinforcement learning (ERL), which introduces natural selection into the reinforcement learning paradigm [95]. They encoded not only action functions which determine EANN's behaviours, but also evaluation functions, in chromosomes consisting of over 280 bits. Changes in evaluations from generation to generation provide reinforcement feedback for driving EANN's (agent's) further learning. In a broader sense, the evolution of evaluation functions could be viewed as part of the evolution of learning rules. Ackley and Littman's experiments shows that evolutionary reinforcement learning is more effective than either of its two components, i.e., evolution and reinforcement learning. Related work on the evolution of evaluation functions includes some carried out in the artificial life area, which is beyond the scope of this paper.

Another topic which we do not try to cover in this paper is the meta-level learning, i.e., learn how to learn learning rules. We have only discussed how to learn connection weights, architectures and learning rules in this paper, but there is always a meta-level, even a meta-meta-level, etc., which could be pursued further, at least theoretically. This topic would be more suitable for a separate paper since the major concern of this paper is biased towards the engineering side, as indicated at the beginning of this paper.

5 Concluding Remarks

Incorporating evolution into connectionist systems is a very active research area. A lot of work has been done in recent years, but there is one important aspect to which not enough attention has been paid, i.e., interactions among different kinds of evolution in a connectionist system. This section first describes a general framework for EANNs which classifies different kinds of evolution into different levels and discusses interactions among these levels, then concludes with a summary.

5.1 A General Framework for EANNs

As can be seen from previous reviews, evolution has been incorporated into ANNs at three different levels, that is, the evolution of connection weights, of architectures, and of learning rules. Different levels of evolution react to different environments and use different time scales. It has been widely accepted that the evolution of connection weights, i.e., learning, proceeds at the lowest level on the fastest time scale [92, 31] in an environment decided by the architecture, the learning rule, and the learning task. There are, however, two alternatives to look at the level of the evolution of architectures and that of learning rules; either the evolution of architectures is at the highest level and that of learning rules at the lower one, or vice versa. The lower the level of an evolution is, the faster the time scale it is on.

From the point of view of engineering, the decision on the level of evolution depends on what kind of prior knowledge is available. If there is more prior knowledge about EANN's architectures than that about their learning rules or a particular class of architectures is pursued, it is better to put the evolution of architectures at the highest level because such knowledge can be encoded in architecture's chromosomal representation to reduce the (architecture) search space and the lower level evolution of learning rules can be more biased towards this kind 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 there is a special interest in certain type of learning rules. However, there is usually little prior knowledge available about both architectures and learning rules in practice, except for some very vague statements [2]. In this case, it might be more appropriate to put the evolution of architectures at the highest level, since the optimality of a learning rule would be easier to evaluate in an environment including the architecture the rule is applied to.

A general framework for EANNs is given in Figure 4 based on our previous work [4, 98]. It can be viewed as a hierarchical adaptive system with three levels. At the highest level (represented by the largest circle), architectures evolve on the slowest time scale in an environment decided by the task to be accomplished by the system. For each architecture, there is a lower level evolution, the evolution of learning rules (represented by the medium circle), associated with it, which proceeds on a faster time scale in an environment decided by the task as well as the architecture. As a result, the learning rule evolved is optimised towards the architecture, not generally applicable to any architectures. For each learning rule, there is an even lower level evolution, the evolution of connection weights (represented by the smallest circle), associated with it, which proceeds on the fastest time scale in an environment decided by the task, the architecture, and the learning rule.

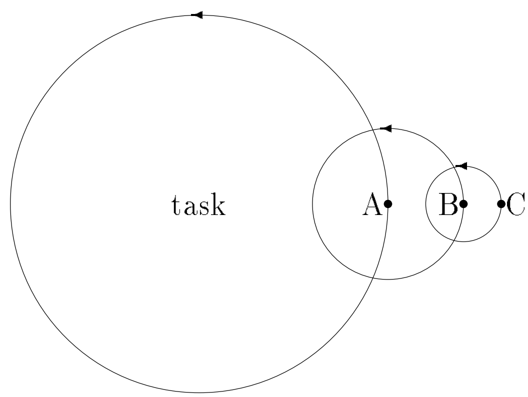

Figure 4: A general framework for EANNs. The size of an circle illustrates the amount of restrictions set on an environment; the larger the size, the less the amount. The largest circle represents the evolution of architectures whose environment is decided solely by the task to be accomplished by the EANN. The smallest circle represents the evolution of connection weights whose environment is constrained by, besides the task, both the architecture and the learning rule at higher levels. An analogy to the solar system could be drawn here. When sets of connection weights (moons) are evolving, where C stands for a particular set (moon), they are in an environment decided by the learning rule B (planet), the architecture A (solar system) and the task (Milky Way galaxy). The optimality of an EANN means the optimal combination of architecture, learning rule and connection weights.

The framework described in Figure 4 could be viewed as a hierarchical model of a general adaptive system if we do not constrain ourselves to GA-based evolutionary search procedures, as stated in the beginning of this paper. Simulated annealing, gradient descent search, evolution strategies, evolutionary programming, and even one-shot (only-one-candidate) search can all be considered as some types of evolutionary search procedures. As a result, the framework provides a basis of comparing various EANN models according to search procedures they used at different levels. A 3-dimensional space where 0 represents one-shot search and $+\infty$ represents exhaustive search along each axis can be defined, and every point in the space corresponds to a particular EANN model.

There are a couple of points worth mentioning about the general framework described by Figure 4. First, although the word *optimal* is normally used, it is very hard in practice to obtain an exact global optimum in a vast and complex space like those considered here. Fortunately, it is often the case in real world applications that a good approximate solution (near optimum) is all that needed, not necessarily an exact global optimum. The criterion of "good enough" varies from problem to problem. The evolutionary process is actually trying to find a near optimal solution, instead of an exact one.

Second, global search procedures, like GAs, are usually computationally expensive. That is way we do not always use GAs at all levels of evolution. It is, however, beneficial to introduce certain kind of global search at a particular level of an EANN, especially when there is little prior knowledge available at that level and the performance of the EANN is required to be high because trial-and-error or other heuristic methods are very inefficient in such circumstance. As the power of parallel computers increases rapidly, the simulation of large EANNs becomes feasible. Such simulation will not only offer a better opportunity to discover novel EANN architectures and learning rules, but also offer a way to model the creative process as a result of EANN's adaptation to a dynamic environment.

5.2 Summary

This paper reviews various efforts made on the combination of evolutionary search procedures with ANNs under a unified framework — EANN. One of the major features of EANNs is their potentiality in discovering adaptively novel architectures and learning rules which are not known before. Three levels of evolution, i.e., the evolution of connection weights, of architectures and of learning rules, are identified and analysed in this paper.

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 tend 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 tend to exploit the optimal regions in fine grain (finding an optimal solution). Such designed EANNs have been shown to be quite competitive in terms of the quality of solutions found and the computational cost.

This paper also describes a general framework for EANNs, which forms a basis for comparing and evaluating different EANN models in the model space. The general framework gives a clearer picture of the role of each kind of evolution and interactions among them, and make the design of a new EANN model easier.

Acknowledgement — A preliminary version of this paper was part of a research proposal supervised by Prof. R. Brent, whose encouragement to the author's work has made this paper possible. The author is also grateful to R. Brent, J. Brotchie, I. Macleod, J. Mashford, B. Marksjö and R. Sharpe for reading and commenting on various versions of this paper. Referees' valuable suggestions and comments have helped to improve the paper greatly.

References

[1] M. Rudnick.
A bibliography of the intersection of genetic search and artificial neural networks.
Technical Report CS/E 90-001, Department of Computer Science and Engineering, Oregon Graduate Institute of Science and Technology, January 1990.

[2] G. Weiss.
Combining neural and evolutionary learning: aspects and approaches.
Technical Report FKI-132-90, Institut für Informatik, Technische Universität München, May 1990.

[3] D. H. Ackley.
A Connectionist Machine for Genetic Hillclimbing.
Kluwer Academic Publishers, Boston, 1987.

[4] X. Yao.
Evolution of connectionist networks.
In T. Dartnall, editor, *Preprints of the Int'l Symp. on AI, Reasoning & Creativity,* pages 49–52, Queensland, Australia, 1991. Griffith University.

[5] D. E. Rumelhart, G. E. Hinton, and R. J. Williams.
Learning internal representations by error propagation.
In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Vol. I*, pages 318–362. MIT Press, Cambridge, MA, 1986.

[6] J. H. Holland.
Adaptation in Natural and Artificial Systems.
University of Michigan Press, Ann Arbor, MI, 1975.

[7] D. E. Goldberg.
Genetic Algorithms in Search, Optimization, and Machine Learning.
Addison-Wesley Publishing Company, Inc., Reading, MA, 1989.

[8] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi.
Optimization by simulated annealing.
Science, 220:671–680, 1983.

[9] X. Yao.
Simulated annealing with extended neighbourhood.
International Journal of Computer Mathematics, 40:169–189, 1991.

[10] T. Bäck, F. Hoffmeister, and H.-P. Schwefel.
A survey of evolution strategies.
In R. K. Belew and L. B. Booker, editors, *Proc. of the Fourth Int'l Conf. on Genetic Algorithms*, pages 2–9. Morgan Kaufmann, San Mateo, CA, 1991.

[11] L. J. Fogel, A. J. Owens, and M. J. Walsh.
Artificial Intelligence Throug Simulated Evolution.
John Wiley & Sons, New York, NY, 1966.

[12] S. E. Fahlman and C. Lebiere.
The cascade-correlation learning architecture.
In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 524–532. Morgan Kaufmann, San Mateo, CA, 1990.

[13] M. Frean.
The upstart algorithm: a method for constructing and training feedforward neural networks.
Neural Computation, 2:198–209, 1990.

[14] M. C. Mozer and P. Smolensky.
Skeletonization: a technique for trimming the fat from a network via relevance assessment.

Connection Science, 1:3–26, 1989.

[15] J. Sietsma and R. J. F. Dow.
Creating artificial neural networks that generalize.
Neural Networks, 4:67–79, 1991.

[16] Y. Hirose, K. Yamashita, and S. Hijiya.
Back-propagation algorithm which varies the number of hidden units.
Neural Networks, 4:61–66, 1991.

[17] Y. LeCun, J. S. Denker, and S. A. Solla.
Optimal brain damage.
In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan Kaufmann, San Mateo, CA, 1990.

[18] P. J. Werbos.
Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.
PhD thesis, Harvard University, Cambridge, MA, 1974.

[19] T. J. Sejnowski and C. R. Rosenberg.
Parallel networks that learn to pronounce English text.
Complex Systems, 1:145–168, 1987.

[20] K. J. Lang, A. H. Waibel, and G. E. Hinton.
A time-delay neural network architecture for isolated word recognition.
Neural Networks, 3:33–43, 1990.

[21] R. P. Gorman and T. J. Sejnowski.
Learned classification of sonar target using a massively-parallel network.
IEEE Trans. on Acoustics, Speech, and Signal Processing, ASSP-36:1135–1140, 1988.

[22] R. S. Sutton.
Two problems with backpropagation and other steepest-descent learning procedures for networks.
In *Proc. of 8th Annual Conf. of the Cognitive Science Society*, pages 823–831. Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

[23] G. E. Hinton.
Connectionist learning procedures.
Artificial Intelligence, 40:185–234, 1989.

[24] D. Whitley and T. Hanson.
Optimizing neural networks using faster, more accurate genetic search.

In J. D. Schaffer, editor, *Proc. of the Third Int'l Conf. on Genetic Algorithms and Their Applications*, pages 391–396. Morgan Kaufmann, San Mateo, CA, 1989.

[25] D. Montana and L. Davis.
Training feedforward neural networks using genetic algorithms.
In *Proc. of Eleventh Int'l Joint Conf. on Artificial Intelligence*, pages 762–767. Morgan Kaufmann, San Mateo, CA, 1989.

[26] T. P. Caudell and C. P. Dolan.
Parametric connectivity: training of constrained networks using genetic algorithms.
In J. D. Schaffer, editor, *Proc. of the Third Int'l Conf. on Genetic Algorithms and Their Applications*, pages 370–374. Morgan Kaufmann, San Mateo, CA, 1989.

[27] D. B. Fogel, L. J. Fogel, and V. W. Porto.
Evolving neural networks.
Biological Cybernetics, 63:487–493, 1990.

[28] P. Bartlett and T. Downs.
Training a neural network with a genetic algorithm.
Technical Report, Dept. of Elec. Eng., Univ. of Queensland, January 1990.

[29] D. Whitley, T. Starkweather, and C. Bogart.
Genetic algorithms and neural networks: optimizing connections and connectivity.
Parallel Computing, 14:347–361, 1990.

[30] J. Heistermann and H. Eckardt.
Parallel algorithms for learning in neural networks with evolution strategy.
In D. J. Evans, G. R. Joubert, and F. J. Peters, editors, *Proc. of Parallel Computing 89*, pages 275–280. Elsevier Science Publishers B.V., Amsterdam, 1989.

[31] R. K. Belew, J. McInerney, and N. N. Schraudolph.
Evolving networks: using genetic algorithm with connectionist learning.
Technical Report #CS90-174 (Revised), Computer Science & Engr. Dept. (C-014), Univ. of California at San Diego, La Jolla, CA 92093, USA, February 1991.

[32] N. J. Radcliffe.
Genetic Neural Networks on MIMD Computers (compressed edition).

PhD thesis, Dept. of Theoretical Phys., University of Edinburgh, Scotland, UK, 1990.

[33] H. de Garis.
Genetic programming.
In *Proc. of Int'l Joint Conf. on Neural Networks*, pages I–194–I–197. Lawrence Erlbaum Associates, Hillsdale, NJ, 1990.

[34] D. Whitley.
The GENITOR algorithm and selective pressure: why rank-based allocation of reproductive trials is best.
In J. D. Schaffer, editor, *Proc. of the Third Int'l Conf. on Genetic Algorithms and Their Applications*, pages 116–121. Morgan Kaufmann, San Mateo, CA, 1989.

[35] N. N. Schraudolph and R. K. Belew.
Dynamic parameter encoding for genetic algorithms.
Technical Report LAUR 90-2795, Center for Nonlinear Studies, Los Alamos National Laboratory, Los Alamos, NM, 1990.

[36] M. Takahashi, M. Oita, S. Tai, K. Kojima, and K. Kyuma.
A quantized back propagation learning rule and its application to optical neural networks.
Optical Computing and Processing, 1:175–182, 1991.

[37] W. Balzer, M. Takahashi, J. Ohta, and K. Kyuma.
Weight quantization in Boltzmann machines.
Neural Networks, 4:405–409, 1991.

[38] M. Hoehfeld and S. E. Fahlman.
Learning with limited numerical precision using the Cascade-Correlation algorithm.
IEEE Trans. on Neural Networks, 3:602–611, 1992.

[39] J. W. L. Merrill and R. F. Port.
A stochastic learning algorithm for neural networks.
Technical Report 236, Dept. of Linguistics and Computer Science, Indiana Univ., Bloomington, IN, 1988.

[40] R. S. Rosenberg.
Simulation of genetic populations with biochemical properties (doctoral dissertation, university of michigan).
Dissertation Abstract International, 28(7):2732B, 1967.

[41] J. D. Schaffer and A. Morishima.
An adaptive crossover distribution mechanism for genetic algorithms.
In *Proc. of 2nd Int'l Conf. on Genetic Algorithms and Their Applications*, pages 36–40. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[42] S. W. Wilson.
Classifier system and the Animate problem.
Machine Learning, 2:199–228, 1987.

[43] L. Davis.
Handbook of Genetic Algorithms.
Van Nostrand Reinhold, New York, NY 10003, 1991.

[44] H. H. Szu and R. L. Hartley.
Nonconvex optimization by fast simulated annealing.
Proceedings of IEEE, 75:1538–1540, 1987.

[45] H. Kitano.
Empirical studies on the speed of convergence of neural network training using genetic algorithms.
In *Proc. of the Eighth Nat'l Conf. on AI (AAAI-90)*. MIT Press, Cambridge, MA, 1990.

[46] X. Yao.
Optimization by genetic annealing.
In M. Jabri, editor, *Proc. of Second Australian Conf. on Neural Networks*, pages 94–97, Sydney, Australia, 1991.

[47] S. E. Fahlman.
Faster-learning variations on back-propagation: an empirical study.
In D. S. Touretzky, G. E. Hinton, and T. J. Sejnowski, editors, *Proc. of the 1988 Connectionist Models Summer School*, pages 38–51. Morgan Kaufmann, San Mateo, CA, 1988.

[48] G. F. Miller, P. M. Todd, and S. U. Hegde.
Designing neural networks using genetic algorithms.
In J. D. Schaffer, editor, *Proc. of the Third Int'l Conf. on Genetic Algorithms and Their Applications*, pages 379–384. Morgan Kaufmann, San Mateo, CA, 1989.

[49] H. Kitano.
Designing neural networks using genetic algorithms with graph generation system.

Complex Systems, 4:461–476, 1990.

[50] S. A. Harp, T. Samad, and A. Guha.
Towards the genetic synthesis of neural networks.
In J. D. Schaffer, editor, *Proc. of the Third Int'l Conf. on Genetic Algorithms and Their Applications*, pages 360–369. Morgan Kaufmann, San Mateo, CA, 1989.

[51] J. D. Schaffer, R. A. Caruana, and L. J. Eshelman.
Using genetic search to exploit the emergent behavior of neural networks.
Physica D, 42:244–248, 1990.

[52] S. W. Wilson.
Perceptron redux: emergence of structure.
Physica D, 42:249–256, 1990.

[53] N. Dodd, D. Macfarlane, and C. Marland.
Optimisation of artificial neural network structure using genetic techniques implemented on multiple transputers.
In *Proceedings of Transputing'91*, 1991.

[54] S. A. Harp, T. Samad, and A. Guha.
Designing application-specific neural networks using the genetic algorithm.
In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 447–454. Morgan Kaufmann, San Mateo, CA, 1990.

[55] W. B. Dress.
Darwinian optimization of synthetic neural systems.
In M. Caudill and C. Butler, editors, *Proc. of the 1st IEEE Int'l Conf. on Neural Networks, Vol. 3*, pages 769–775. IEEE, New York, NY, 1987.

[56] A. Bergman and M. Kerszberg.
Breeding intelligent automata.
In M. Caudill and C. Butler, editors, *Proc. of the 1st IEEE Int'l Conf. on Neural Networks, Vol. 3*, pages 63–69. IEEE, New York, NY, 1987.

[57] P. J. B. Hancock.
GANNET: desgin of a neural net for face recognition by genetic algorithm.
Technical Report CCCN-6, Center for Cognitive and Computational Neuroscience, Dept. of Computing Sci. and Psychology, Stirling University, Stirling FK9 4LA, UK, August 1990.

[58] C. P. Dolan and M. G. Dyer.
Towards the evolution of symbols.

In *Proc. of 2nd Int'l Conf. on Genetic Algorithms and Their Applications*, pages 123–131. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[59] D. Whitley and C. Bogart.
The evolution of connectivity: pruning neural networks using genetic algorithms.
In *Proc. of Int'l Joint Conf. on Neural Networks*, pages I–134–I–137. Lawrence Erlbaum Associates, Hillsdale, NJ, 1990.

[60] B. Mariĉić and Z. Nikolov.
GENNET — system for computer aided neural network design using genetic algorithms.
In *Proc. of Int'l Joint Conf. on Neural Networks*, pages I–102–I–105. Lawrence Erlbaum Associates, Hillsdale, NJ, 1990.

[61] D. G. Stork, S. Walker, M. Burns, and B. Jackson.
Preadaption in neural circuits.
In *Proc. of Int'l Joint Conf. on Neural Networks*, pages I–202–I–205. Lawrence Erlbaum Associates, Hillsdale, NJ, 1990.

[62] S. Bornholdt and D. Graudenz.
General asymmetric neural networks and structure design by genetic algorithms.
Technical Report DESY 91-046, Deutsches Elektronen — Synchrotron, Notkestrasse, D-2000 Hamburg 52, May 1991.

[63] F. Wong, P. Tan, and X. Zhang.
Neural networks, genetic algorithm and fuzzy logic for forecasting.
In *Proc. of the Third Int'l Conf. on Advanced Trading Technologies — AI Applications on Wall Street and Worldwide*, New York, NY, To appear, 1992.

[64] G. Mani.
Learning by gradient descent in function space.
In *Proc. of IEEE Int'l Conf. on System, Man, and Cybernetics*, pages 242–247, Los Angeles, CA, 1990.

[65] D. R. Lovell and A. C. Tsoi.
The performance of the Neocognitron with various S-cell and C-cell transfer functions.
Intelligent Machines Lab., Dept. of Elec. Eng., Univ. of Queensland, April 1992.

[66] M. Bichsel and P. Seitz.
Minimum class entropy: a maximum information approach to layered networks.
Neural Networks, 2:133–141, 1989.

[67] D. B. Fogel.

An information criterion for optimal neural network selection.
IEEE Trans. on Neural Networks, 2:490–497, 1991.

[68] J. Utans and J. Moody.
Selecting neural network architectures via the prediction risk: application to corporate bond rating prediction.
In *Proc. of the First Int'l Conf. on AI Applications on Wall Street*. IEEE Computer Society Press, Los Alamitos, CA, 1991.

[69] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth.
Occam's razor.
Information Processing Letters, 24:377–380, 1987.

[70] J. Rissanen.
Modeling by shortest data description.
Automatica, 14:465–471, 1978.

[71] E. Mjolsness, D. H. Sharp, and B. K. Alpert.
Scaling, machine learning, and genetic neural nets.
Advances in Applied Mathematics, 10:137–163, 1989.

[72] J. W. L. Merrill and R. F. Port.
Fractally configured neural networks.
Neural Networks, 4:53–60, 1991.

[73] B. Lewin.
Units of transcription and translation: sequence components of heterogeneous nuclear RNA and messager RNA.
Cell, 4, 1975.

[74] E. R. Kandel and J. H. Schwartz.
Principles of Neural Science (2nd Ed.).
Elsevier Science Publishers B.V., New York, NY, 1985.

[75] S. A. Harp and T. Samad.
Genetic synthesis of neural network architecture.
In L. Davis, editor, *Handbook of Genetic Algorithms*, pages 203–221. Van Nostrand Reinhold, New York, NY 10003, 1991.

[76] Y. Doi.
Morphogenesis of Life Forms.
Saiensu-sha, 1988.

[77] B. West.
The fractal structure of the human lung.

In S. Kelso, editor, *Proc. of the Conf. on Dynamic Patterns in Complex Systems.* Ehrlbaum, New York, NY, 1988.

[78] G. M. Edelman.
Neural Darwinism: The Theory of Neuronal Group Selection.
Basic Books, New York, NY, 1987.

[79] R. A. Jacobs.
Increased rates of convergence through learning rate adaptation.
Neural Networks, 1:295–307, 1988.

[80] D. J. Chalmers.
The evolution of learning: an experiment in genetic connectionism.
In D. S. Touretzky, J. L. Elman, and G. E. Hinton, editors, *Proceedings of the 1990 Connectionist Models Summer School*, pages 81–90. Morgan Kaufmann, San Mateo, CA, 1990.

[81] Y. Bengio and S. Bengio.
Learning a synaptic learning rule.
Technical Report 751, Départment d'Informatique et de Recherche Opérationelle, Université de Montréal, Canada, November 1990.

[82] S. Bengio, Y. Bengio, J. Cloutier, and J. Gecsei.
On the optimization of a synaptic learning rule.
In *Preprints of the Conference on Optimality in Artificial and Biological Neural Networks*, 1991.

[83] J. F. Fontanari and R. Meir.
Evolving a learning algorithm for the binary perceptron.
Network, 2:353–359, 1991.

[84] D. O. Hebb.
The Organization of Behavior: A Neurophysiological Theory.
John Wiley & Sons, New York, NY, 1949.

[85] P. J. B. Hancock, L. S. Smith, and W. A. Phillips.
A biologically supported error-correcting learning rule.
In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Proc. of Int'l Conf. on Artificial Neural Networks — ICANN-91, Vol. 1*, pages 531–536. Elsevier Science Publishers B.V., Amsterdam, 1991.

[86] A. Artola, S. Broecher, and W. Singer.
Different voltage-dependent thresholds for inducing long-term depression and long-term potentiation in slices of rat visual cortex.

Nature, 347:69–72, 1990.

[87] J. M. Smith.
When learning guides evolution.
Nature, 329:761–762, 1987.

[88] G. E. Hinton and S. J. Nowlan.
How learning can guide evolution.
Complex Systems, 1:495–502, 1987.

[89] R. K. Belew.
Evolution, learning and culture: computational metaphors for adaptive algorithms.
Technical Report #CS89-156, Computer Science & Engr. Dept. (C-014), Univ. of California at San Diego, La Jolla, CA 92093, USA, September 1989.

[90] S. Nolfi, J. L. Elman, and D. Parisi.
Learning and evolution in neural networks.
Technical Report CRT-9019, Center for Research in Language, University of California, San Diego, La Jolla, CA 92093-0126, U.S.A., July 1990.

[91] R. Keesing and D. G. Stork.
Evolution and learning in neural networks: the number and distribution of learning trials affect the rate of evolution.
In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems (3)*, pages 804–810. Morgan Kaufmann, San Mateo, CA, 1991.

[92] H. Mühlenbein and J. Kindermann.
The dynamics of evolution and learning — towards genetic neural networks.
In R. Pfeifer et al., editor, *Connectionism in Perspective*, pages 173–198. Elsevier Science Publishers B.V., Amsterdam, 1989.

[93] H. Mühlenbein.
Adaptation in open systems: learning and evolution.
In J. Kindermann and C. Lischka, editors, *Workshop Konnektionismus*, pages 122–130. GMD, Postfach 1240, D-5205 St., Augustin, Germany, 1988.

[94] J. Paredis.
The evolution of behavior: some experiments.
In J.-A. Meyer and S. W. Wilson, editors, *Proc. of the First Int'l Conf. on Simulation of Adaptive Behavior: From Animals to Animats*. MIT Press, Cambridge, MA, 1991.

[95] D. H. Ackley and M. S. Littman.
Learning from natural selection in an artificial environment.
In *Proc. of Int'l Joint Conf. on Neural Networks*, pages I–189–I–193. Lawrence Erlbaum Associates, Hillsdale, NJ, 1990.

[96] B. Widrow and M. E. Hoff.
Adaptive switching circuits.
In *1960 IRE WESTCON Convention Record*, pages 96–104. IRE, New York, NY, 1960.

[97] D. Parasi, F. Cecconi, and S. Nolfi.
Econets: neural networks that learn in an environment.
Network, 1:149–168, 1990.

[98] X. Yao.
Evolution of connectionist networks.
In T. Dartnall, editor, *AI and Creativity*. Kluwer Academic Publishers, Boston, to appear.