11

Genetic Algorithms and Neural Networks

D. WHITLEY

11.1 INTRODUCTION

Genetic algorithms and neural networks are both inspired by computation in biological systems. A good deal of biological neural architecture is determined genetically. It is therefore not surprising that as some neural network researchers explored how neural systems are organized that the idea of evolving neural architectures should arise.

Genetic algorithms have been used in conjunction with neural networks in three major ways. First, they have been used to set the weights in fixed architectures. This includes both supervised learning applications and reinforcement learning applications. In related work, a genetic algorithm has been used to set the learning rates which in turn are used by other types of learning algorithms. Genetic algorithms have also been combined with more traditional forms of gradient based search.

Second, genetic algorithms have been used to learn neural network topologies. When evolving neural networks topologies for function approximation, this includes the problem of specifying how many hidden units a neural network should have and how the nodes are connected.

A third major application is the use of genetic algorithms to select training data and to interpret the output behavior of neural networks.

Schaffer, Whitley and Eshelman (1992) survey these various areas in an introduction to the proceeding of a 1992 workshop on *Combinations of Genetic algorithms and Neural Networks.* The current paper is tutorial in nature and highlights select cases and briefly references some of the work that has been introduced in the last 3 years.

Genetic Algorithms in Engineering and Computer Science
Editor J. Periaux and G. Winter

11.2 GENETIC ALGORITHMS FOR PREPROCESSING AND INTERPRETING DATA

Two examples of using genetic algorithms for preprocessing data is given in the work of Chang and Lippmann (1991) and the work of Brill, Brown and Martin (1992). In both cases, a large number of inputs were available as input to a K nearest neighbor (KNN) classifier. In this case the coding can be a simple binary string indicating whether a particular input or combination of inputs can be deleted from the input set without significantly changing the classification behavior. In the Chang and Lippmann application, the genetic algorithm was able to reduce the input set from 153 to 33 input features. Brill, Brown and Martin also were able to reduce the input set, but their goal was not just to reduce the set of inputs to the nearest neighbor classifier, but to also identify inputs that would also work well for a counterpropagation network. The nearest neighbor classifier was used for feature selection since the evalutation of a feature set is much faster with the KNN classifier than the counterpropagation network. Nevertheless, the reduced input set for the KNN classifier also worked well for the counterpropagation network.

Genetic algorithms have not only been used to reduce the input data set but also to interpret outputs of a neural network. Eberhart and Dobbins (1991; Eberhart 1992) used a genetic algorithm to search for the decision surface that identified boundary cases of appendicitis as predicted by a neural network. For example, what inputs lead to a classification of 0.5, where 0.5 indicates a borderline case, i.e., a case that lies on the boundary between the decision regions that classify cases as positive or negative examples of appendicitis? It can also be useful to determine what are considered to be what Eberhart calls 'quintessential' examples of appendicitis as predicted by a neural network. In this case, what inputs lead to a classification of 1.0, where 1.0 corresponds to a classic case of appendicitis?

Asking for an input that yields an output of 1.0 or 0.5 is really a form of network inversion; in other words, this is analogous to running the neural network backwards. One can literally attempt to run a neural network backwards by using backpropagation to look for hidden node and input node activations that yield a particular output, but the process can be time consuming and does not always work well since the classification of a neural network is often many-to-one and not an invertible function. Eberhart and Dobbins simply searched the input space for strings that produced the desired output. By running a genetic algorithm multiple times they were able to obtain multiple patterns that mapped to a particular output.

Such information can be used in two ways. First, it can be used as an explanation tool. Knowing quintessential examples as well as borderline cases can help explain how a network classifies novel inputs. Second, it can also be used to assess what a neural network has learned and whether the cases that it considers to be quintessential and borderline are reasonable.

11.3 GENETIC ALGORITHMS FOR TRAINING NEURAL NETWORKS

The idea of training neural networks with genetic algorithms can be found in Holland's 1975 book *Adaptation in Natural and Artificial Systems*. Most of the actual work in this area is far more recent. Belew, McInernery and Schraudolph (1990), Harp, Samad and Guha (1989;1990) and Schaffer, Caruana and Eshelman (1990) all used genetic algorithm to set the learning and momentum rates for feedforward neural networks. Mühlenbein also contributed to the early efforts in this area (1990; Mühlenbein and Kindermann, 1989).

This tuning was often done in conjunction with other changes to the network, such as weight initialization or changing the network topology. In addition, there have also been several researchers that attempted to train feedforward neural networks for decision problems using genetic algorithms (Whitley and Hanson, 1989; Montana and Davis, 1989; Whitley et al. 1990). Related to this is the use of genetic search in the optimization of Kanerva's (1988) sparse distributed memories by Rogers (1990) and Wilson's work (1990) which learned predicates over input features to construct new higher order inputs to a perceptron.

Rogers (1990) has used genetic algorithms to optimize the "location addresses" (i.e. the layer mapping inputs to hidden units) of a sparse distributed memory. Das and Whitley (1992) extend the work of Rogers by using a genetic algorithm for "location address" optimization that actively extracts information about multiple local minima based on relative global competitiveness. Each local optimum in this particular definition of the search space represents a different and distinct data pattern that correlates with some output or event of interest. This allows multiple data patterns to be tracked simultaneously, where each pattern corresponds to a different local optimum in location address space.

The application of genetic algorithms to simple weight training for neural networks has been hampered by two factors. First, gradient methods have been developed that are highly effective for weight training in supervised learning applications where input-output training examples are available and where the target network is a simple feed forward network. Second, the problem of training a feed forward Artificial Neural Network (ANN) may represent an application that is inherently not a good match for genetic algorithms that rely heavily on recombination. Some researchers do not use recombination (e.g. Porto and Fogel, 1990) while other have used small populations and high mutation rates in conjunction with recombination. We first look at why optimizing the weights in a neural network may cause problem for algorithms that rely heavily on simple recombination schemes.

11.3.1 The Problem With ANN

One reason that genetic algorithms may not yield a good approach to optimizing neural network weights is the *Competing Conventions Problem*. Nick Radcliffe (1990; 1991) has also named this the *Permutations Problem*. The source of the problem is that there can be numerous equivalent symmetric solutions to a neural network weight optimization problem.

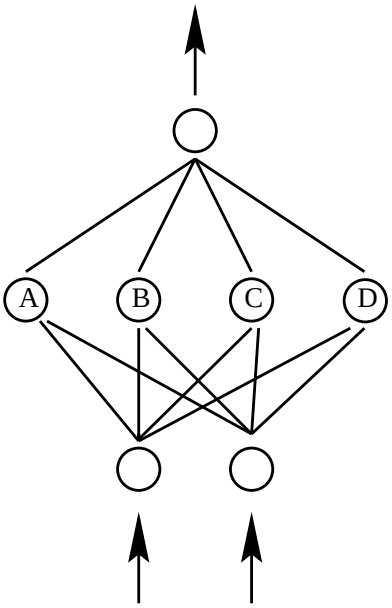

Figure 11.1 A simple feedforward neural network. Note that rearranging the positions of the hidden units does not change the functionality of the network.

Figure 11.1 illustrates a simple feedforward network. Assume that the vector

$$w_{a,1}, w_{a,2}, w_{a,3}, w_{b,1}, w_{b,2}, w_{b,3}, w_{c,1}, w_{c,2}, w_{c,3}, w_{d,1}, w_{d,2}, w_{d,3}$$

is an arbitrary assignment of weights to this neural network, where $w_{\alpha,i}$ passes through hidden node α and $i = 1, i = 2$ are input connections and $i = 3$ is the output connection. Note that for every vector of this form there are $4! = 24$ equivalent vectors representing exactly the same solution. All permutations over the set of hidden unit indices, $\{a, b, c, d\}$, are equivalent vectors in terms of neural network functionality and in terms of the resulting evaluation function. This is because rearranging the order of the hidden units has no effect on the functionality of the network. Thus, given H hidden units in a simple fully connected feedforward network, there are $H!$ symmetries and up to $H!$ equivalent solutions.

The problem this creates for a genetic algorithm that uses simple recombination is as follows. If one does simple crossover on a permutation such as [A B C D] and [D A C B] then the offspring will duplicate some elements of the permutation and will omit others. Similarly, if different strings try to map functionality of hidden nodes in different ways, then recombining these strings will result in duplication of some hidden units and omission of other hidden units. In this case, using a population-based form of search can be a disadvantage, since different strings in the population may not map functionality to the different hidden units in the same way.

Various solutions have been proposed to the Competing Conventions Problems.

Early on, Montana and Davis (1989) attempted to identify functional aspects of hidden units during recombination in order to perform a type of intelligent crossover. Radcliffe (1991) also suggested a solution whereby hidden units are treated as a multiset: hidden units with the same connectivity are considered to be the same, but hidden units might have different connectivities. During recombination, one can search through the hidden units to determine which are identical and use this information to guide crossover. Hancock (1992) has implement this idea as well as extensions to consider how similar hidden units are; he concludes the permutation problem is not as bad as has often been suggested. More recently, Korning (1994) has suggested that the traditional use of the standard quadratic error measurement, $(target - observed)^2$, is part of the problem and suggests the use of other fitness measurements. Korning also suggests "killing off" any offspring that do not meet minimal fitness requirements, which might filter out offspring from incompatible parents. Overall however, it is very difficult to find cases where genetic algorithms have been shown to yield results better than gradient based methods for supervised learning applications.

One recent report returns to a theme initially put forward by Belew et al. (1990). Part of the traditional wisdom (folklore?) which has grown up around genetic algorithms is that a genetic algorithm is good at roughly characterizing the structure of a search space and finding regions of good average fitness, but not adept at exploiting local features of the search space. One way to use a genetic algorithm then is to use it to find an initial set of good weights and then to turn the search over to a gradient based method. Skinner and Broughton (1995) have reported good results with this kind of approach and suggests this method is better than using gradient methods alone for complex problems involving large weight vectors.

11.3.2 Genetic and Evolutionarly Algorithms for Reinforcement Learning

Another stratgey is to use genetic and evolutionary algorithms for weight optimization in domains where gradient methods cannot be directly applied, or where gradient methods are less effective than in simple supervised learning applications. One such application is the use of evolutionary algorithms to train neural networks for reinforcement learning problems and neurocontrol applications. Some results suggest that evolutionary algorithms can be quite competitive against other algorithms that are applicable to reinforcement learning problems. For reinforcement learning applications the set of target outputs that correspond to some set of inputs used to train the net are not known a priori. Rather, the evaluation of the network is performance based. Most existing algorithms attempt to convert the reinforcement learning problem to a supervised learning problem by indirectly or heuristically generating a target output for each input. Some approaches compute an inverse of a system model. The system model maps inputs (current state and control actions) to outputs (the subsequent state). Given a target state, the inverse of the system model can be used to generate actions, which can then be used as a output target (the appropriate action) for a separate controller. This general description is applicable to methods such as "Back propagation through time." "Adaptive critic" methods use a separate evaluation net that learns to predict or evaluate performance at each time step. The prediction can then be used to heuristically generate output targets for an "action" net which controls system behavior. Note, however, that both of

these methods compute target outputs and the resulting gradients either indirectly or heuristically.

Genetic algorithms can be directly applied to reinforcement learning problems because genetic algorithms do not use gradient information, but rather only a relative measure of performance for each set of weight vectors that is evaluated. Genetic algorithms and evolutionary algorithms have been successfully applied to training neural nets to controlling an inverted pendulum. Weiland (1990; 1991) for example trained recurrent networks to balance two inverted pendulums of different lengths at the same time, as well as a jointed pendulum. These algorithms often use smaller population sizes and higher mutation rates to cope with the "Competing Conventions Problems." Whitley et al. (1991; 1993) compared a genetic hill-climber to the well known work of Anderson (1989) which uses the "temporal difference method" (Sutton 1988) to train an "Adaptive Heuristic Critic" (AHC) which in turn is used to generate target outputs for doing reinforcement backpropagation. The results suggests that that the genetic algorithms produced training times comparable to the AHC with reinforcement backpropagation, while generalization was better for the genetic algorithm. .

Whitley et al. (1991; 1993) have argued that comparisons of algorithms for reinforcement learning (and other decision problems) should not only consider learning time but also generalization. Algorithms that learn very quickly can potentially fail to produce an adequate generalized model of the process being learned. Thus, fast learning is not in and of itself a good measurement for evaluating a training algorithm. Generalization is also effected by how the evaluation function is constructed. In reinforcement learning and control problems, the number of possible initial states can be intractable. Thus, evaluation involves sampling the set of possible start states. Evaluation based on a single fixed start state can result in fast learning, but very poor generalization. Evaluation based on a single random start state is somewhat better, but the resulting evaluation is noisy and it difficult to compare the evaluation of one string against another. Evaluation based on a set of start states that uniformly samples the input space would appear to be the best strategy.

11.4 GENETIC ALGORITHMS FOR CONSTRUCTION NEURAL NETWORKS

Some of the early efforts to encode neural network architectures assume that the number of hidden units was bounded; the genetic algorithm could then be used to determine what combinations of weights or hidden units yield improved computational behavior within a finite range of architectures. These directly coded network architectures have usually been trained using back propagation. A common fitness measurement is the training time. Miller and Todd (1989) have explored these ideas, as have Belew, McInerney and Schraudolf (1990). Whitley, Starkweather and Bogart (1990) show that the genetic algorithm can be used to find network topologies that consistently display improved learning speeds over the typical fully connected feed forward network. They also explore how to create selective pressure toward smaller nets and to reduce training time by initializing the reduced networks using weights that have already been optimized for larger fully connected networks.

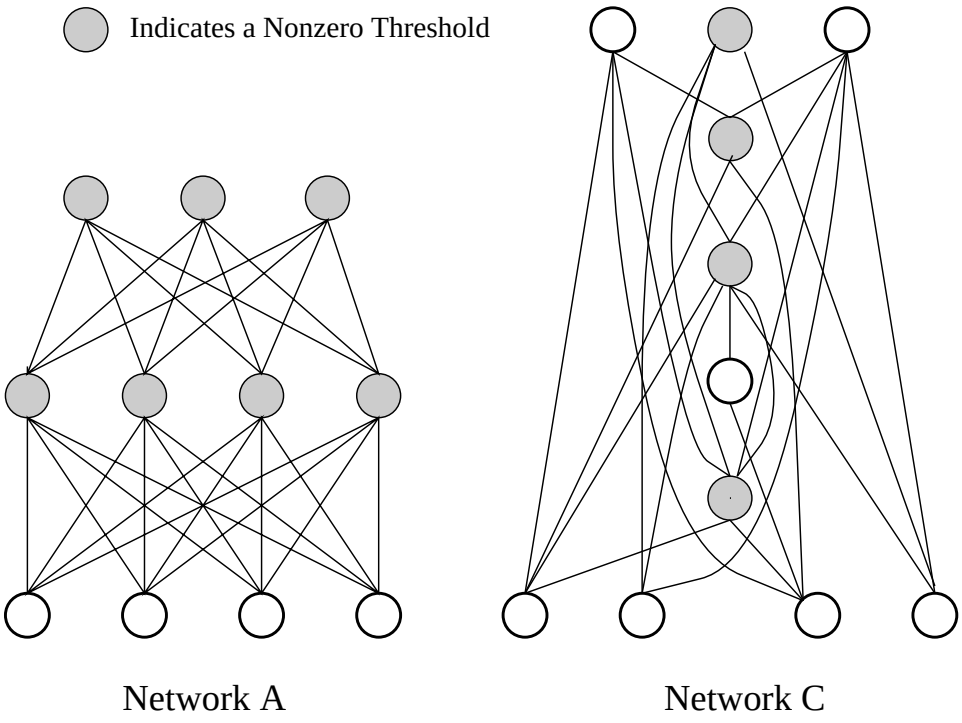

Figure 11.2 A standard feedforward networks for adding two 2-bit numbers and an architecture evolved using a genetic algorithm. The special architecture learns much faster.

An example of the effort to reduce the network topology for a 2-bit adder is given in Figure 11.2. Network C was evolved by a genetic algorithm and learned to add in between 8,000 and 9,000 training epochs on 50 out of 50 tests. Network A failed to converge on 5 of the 50 tests, and over half of the networks required more than 50,000 training epochs to train. A Network B was created by adding direct connections to the input-output nodes of Network A. Network B learned the training set in between 10,000 and 50,000 training epochs on 46 out of 50 tests.

Such early results were encouraging, but the difficulty with directly optimizing a network architecture is the high cost of each evaluation. If we must run a back-propagation algorithm (or some faster, improved form of gradient descent) for each evaluation, the number of evaluations needed to find improved network architectures quickly becomes computationally prohibitive. The computation cost is typically so high as to make genetic algorithms impractical except for optimizing small topologies. For example, if an evaluation function for a modest-sized neural network architecture on a complex problem involves one hour of computation time, then it requires *one year* to do only 9,000 architecture evaluations. If the architecture is complex then 9,000 evaluations is most likely inadequate for genetic search to be effective.

Also, one trend in neural networks that partially addresses the architecture issue is constructive algorithms such as the Cascade Correlation Learning Architecture, which incrementally adds hidden units to the neural network as it learns. Thus, the basis for comparison is not just simple fully connected feedforward networks.

Another more recent effort to evolve neural networks is the work of Angeline et al (1994). The *GeNeralized Acquistion of Recurrent Links*, or GNARL system, uses selection and mutation to search the space of possible recurrent neural network architectures. GNARL attempts to learn weights and topology at the same time. This type of approach differs from constructive algorithms such as Cascade Correlation in that the space of possible architectures is explored in a nonmonotonic fashion.

11.4.1 Neurogenesis: Growing Neural Networks

In the last 5 years some of the most advanced work for using genetic algorithms to develop neural network have focused on growing neural network. Weights and architectures are often developed together. This can include systems such as GNARL. Other researchers have also looked at genetic programming as a way of developing architectures and weights together (Koza and Rice 1991).

Grammar based architecture descriptions have been explored by Kitano (1990), Mjolsness et al. (1988) and by Gruau (1992). Nolfi et al. (1990) have also looked at grammar based systems that retain may of the characteristics of L-systems. By optimizing grammar structures that generate network architectures instead of directly optimizing architectures this research hopes to achieve better scalability, and in some sense, reusability of network architectures. In other words, the goal of this research is to find rules for generating networks which will be useful for defining architectures for some general class of problems. In particular, this would allow developers to define neural structures for smaller problems that could reused as as building blocks for solving larger problems.

One of the earliest efforts to look at network growth was by Mjolsness et al., (1988) which defined a recursive equation for a matrix from which a family of integer matrices could be derived, and then a family of weighted neural nets. The search space is defined over the set of equation coefficients. Mjolsness uses simulated annealing instead of the genetic algorithm to search this space.

Kitano (1990) uses a grammar to generate a family of matrices of size 2^k. The element of the matrix are characters in a finite alphabet. In order to develop matrix M_{k+1} each character of the matrix M_k is replaced by a 2×2 matrix. This connectivity matrix describes the architecture of a neural net. To produce an acyclic graph for a feed forward neural network, only the upper right triangle of the matrix is used.

More recently, Kitano has presented a simple model of neurogenesis that is more biological in nature. In this approach, "axons grow while cell metabolism are being computed." (1995:81). Cell membranes are also modeled that are capable of chemical transport and diffusion. This work appears to be focused on understanding the emergent properties of this type of system.

Gruau (1992) directly develops a cellular development model for growing neural nets called *cellular encoding*. Each cell has a duplicate copy of the "genetic code." Each cell reads the code at a different position. Depending on what is read, a cell can divide, change internal parameters, and finally become a neuron. Arguably, the resulting

language can describe networks in a more elegant and compact way than matrix representations, and the representation can be readily recombined by the genetic algorithm. Gruau used a genetic algorithm to recombine grammar trees representing cellular encodings and has showed that neural networks for the parity problem and symmetry problem could be found. More recently Gruau (1995) has also evolved controls for a 6 legged robot and Whitley, Gruau and Pyeatt (1995) evolve recurrent neurocontrollers for balancing 1 and 2 poles without velocity input information.

11.4.2 A Review of Cellular Development.

Each cell carries a copy of the genetic code in the form of a grammar tree. Each cell also has a pointer which points to a node into the grammar tree. Each node is a program instruction. Development starts with a single *ancestor cell* with connections to input cells and output cells.

In a *Sequential* divide, denoted by S, the parent cell splits into two cells such that the first child inherits all of the input connections of the parent and the second child inherits all of the output connections of the parent; the first child is also connected by a single connection to the second child. In Figure 11.3, during a Sequential divide the second child is placed under the first child. An S node is also a branch point, with the top child cell moving its pointer to the left branch node below S and the bottom child moving its pointer to the right branch node below S.

In a *Parallel* divide, denoted by P, the parent cell splits into two cells that inherit all of the input and output connects of the parent. In Figure 11.3, during a Parallel divide the two child cells are place side by side. A P node is also a branch point, with the left hand child cell moving its pointer to the left branch node below P and the right hand child moving its pointer to the right branch node below P.

The next symbol encountered in Figure 11.3 is the E, which is the *end* or termination symbol. A cell terminates development after reading the E symbol.

The program-symbol **A** increments the threshold of the hidden unit. The program-symbol denoted "-" sets the weight of the input link pointed by the link register to -1. In this example the link register has not been reset and so has its original default setting such that it points to the leftmost fan-in connection.

Figure 11.3 shows an example of a simple grammar tree that generates a XOR networks.

In order to reuse subcomponents of the neural network, cellular encoding uses a special recurrent program-symbol denoted **R**. Associated with **R** is a counter than controls the number of recursive jumps that can be made. When **R** is encountered by a cell, the cell moves its reading head back to the root cell of the grammar tree. The associated counter decrements each time the recursive jump is made. When the counter equals 0 the cell does not reset it pointer, but rather moves forward in the grammar tree, or gives up its reading head and terminates development. Gruau and Whitley (1993) provide an example of how the solution to the XOR net can be generalized to cover all parity problems by placing an **R** symbol in the leftmost leaf node of the grammar tree in Figure 11.3. On parity and symmetry problems, after the genetic algorithm has generated a family of recursively developed networks that handle the lower order cases (3 to 6 inputs), the recursive network encoding represents a general relation and automatically generalizes to handle arbitrarily large problems.

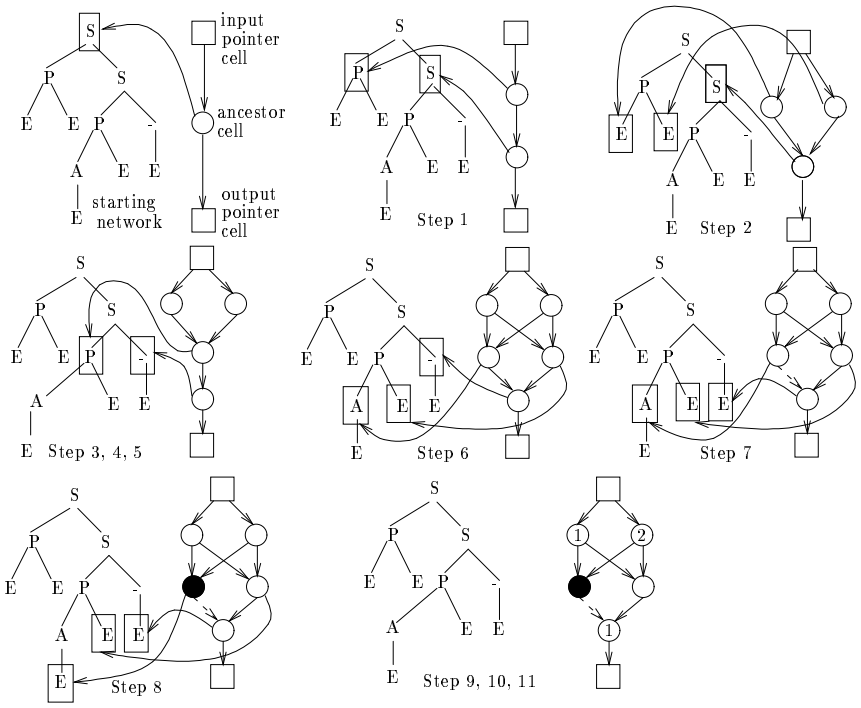

Figure 11.3 The cellular development process. In step 1 the ancestor cell does a sequential divide into 2 cells. In step 2 the uppermost cell from the previous step does a parallel divide. The two cells that are created both read termination symbols in steps 3 and 4; in step 5 the sequential divide is executed. In step 6 a parallel divide is executed. In step 7 the "-" symbol has been executed and a negative weight is introduced feeding into the output node. In step 8 the black cell has changed its threshold. In the final steps, the remaining cells just read termination symbols. (This figure is taken from Gruau and Whitley, 1993).

Another way to reuse development code is to use a form of Automatic Function Definition like that used in Genetic Programming. Subtrees are created, such that the main tree can jump to a subtree, execute the subtree, then return to the associated program-symbol in the main grammar tree. Subtrees thus function like program subroutines. Gruau has used Automatic Function Definition to evolve a mechanism to control the gait of a 6-legged robot. The use of Automatic Function Definition results in simpler, more modular and well structured neural network (Gruau 1995).

11.5 Evolution, Learning and the Baldwin Effect

There has been considerable interest recently in the idea that learning can impact evolution even if learned behaviors are not coded back on the chromosome, as in Lamarckian evolution. The work of Hinton and Nowlan (1987) explains how learning can reshape the fitness landscape, since an individual's fitness is made up of both their genetically determined behavior and learned behavior. If learned behavior has a significant impact on fitness and if the contribution of the learned behavior is stable over time, there can be a selective advantage to having a genetic predisposition that makes it easier to acquire this learned behavior, and eventually, perhaps even to add the behavior to the individual's genetically determined behaviors. Note that this can occur without Lamarckian mechanisms, since there is selection presssure for the learned behavior which can be exploited by Darwinian selection. This idea dates back to Baldwin (1896) and hence is known as the Baldwin Effect.

Such interactions in learning and evolution have been observed when training neural networks using genetic algorithms. Also, the idea of using learning on top of genetic search to speed up the search process has also been explored. Some researchers that explore the interaction of learning and evolution in neurogenetic systems include Ackley and Littman (1991), Gruau and Whitley (1991) and Belew (1989).

11.6 CONCLUSIONS

The challenge facing researchers interested in combinations of genetic algorithms and neural networks is to show how genetic algorithms can make a positive and competitive contribution in the neural networks arena. Currently, it appears that using genetic algorithms to find a set of initial weights before applying gradient based methods may be advantageous for supervised learning classification problems. The application of genetic methods to the development of neural networks for reinforcement learning application also appears to be a worthwhile area for future work. Combinations of genetic algorithms and neural networks are likely to also continue to impact the field of artificial life.

References

Ackley D.H. and Littman M. (1991) Interactions between learning and evolution. In *Proc. of the 2nd Conf. on Artificial Life*, C.G. Langton, ed., Addison-Wesley, 1991.

Anderson C. W. (1989) Learning to Control an Inverted Pendulum Using Neural Networks. *IEEE Control Systems Magazine*, 9, 31-37.

Angeline P.J., Saunders G. M. and Pollack J.B. (1994) An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks* 5(1):54-64.

Baldwin J.M. (1896) A new factor in evolution. *American Naturalist*, 30:441-451, 1896.

Belew R. (1989) When both individuals and populations search: Adding simple learning to the genetic algorithm. In J.D. Schaffer (Ed.), *Third international conference on genetic algorithms* (pp. 34-41). San Mateo, CA: Morgan Kaufmann.

Belew R., McInerney J. and Schraudolph N. (1990) Evolving Networks: Using the Genetic Algorithms with Connectionist Learning. CSE Technical Report CS90-174, Computer Science, UCSD.

Brill F.Z., Brown D.E. and Martin W.N. (1992) Fast genetic selection of features for neural network classifiers. *IEEE Transactions on Neural Networks, 3* (2), 324-328.

Chang E.J. and Lippmann R.P. (1991) Using genetic algorithms to improve pattern classification performance. In R.P. Lippmann, J.E. Moody and D.S. Touretsky (Eds.), *Advances in neural information processing 3* (pp. 797-803). San Mateo, CA: Morgan Kaufmann.

Das R. and Whitley D. (1992) Genetic Sparse Distributed Memories. *Combinations of Genetic Algorithms and Neural Networks.* D. Whitley and J.D. Schaffer (eds.) IEEE Computer Society Press.

Eberhart R.C. and Dobbins R.W. (1991) Designing neural network explanation facilities using genetic algorithms. *IEEE international joint conference on neural networks* (pp. 1758-1763). Singapore: IEEE.

Eberhart R.C. (1992) The role of genetic algorithms in neural network query-based learning and explanation facilities. In *Combinations of Genetic Algorithms and Neural Networks.* D. Whitley and J.D. Schaffer (eds.) IEEE Computer Society Press. Fahlman S. and Lebiere C. (1990). The Cascade Correlation Learning Architecture. In D. Touretzky (Ed), *Advances in Neural Information Processing Systems 2*, Morgan Kaufmann.

Gruau F. (1992) Genetic synthesis of Boolean neural networks with a cell rewriting developmental process. In, *Combination of Genetic Algorithms and Neural Networks*, D. Whitley and J.D. Schaffer, eds, IEEE Computer Society Press, 1992.

Gruau F. and Whitley D. (1993) Adding Learning to the Cellular Development of Neural Networks: Evolution and the Baldwin Effect. *Evolutionary Computation* 1(3): 213–233.

Gruau F. (1995). Automatic Definition of Modular Neural Networks, *Adaptive Behavior*,3(2):151-183.

Hancock P.J.B. (1992) Genetic algorithms and permutation problems: a comparison of recombination operators for neural structure specification. In *Combinations of Genetic Algorithms and Neural Networks.* D. Whitley and J.D. Schaffer (eds.) IEEE Computer Society Press.

Harp S.A., Samad T. and Guha A. (1989) Towards the genetic synthesis of neural networks. In J.D. Schaffer (Ed.), *Third international conference on genetic algorithms* (pp. 360-369).

San Mateo, CA: Morgan Kaufmann.

Harp S.A., Samad T. and Guha A. (1990) Designing application-specific neural networks using the genetic algorithm. In D.S. Touretsky (Ed.), *Advances in neural information processing 2* (pp. 447-454). San Mateo, CA: Morgan Kaufmann.

Hinton G.E. and Nowlan S.J.(1987) How learning can guide evolution. *Complex Systems*, 1:495–502.

Holland J. (1975) *Adaptation in Natural and Artificial Systems.* Ann Arbor, Univ. of Michigan Press.

Kanerva Pentti (1988). *Sparse Distributed Memory.* Cambridge, Mass: MIT Press.

Kitano H. (1990) Designing neural network using genetic algorithm with graph generation system. *Complex Systems*, 4:461–476.

Kitano H. (1995) A simple model of neurogenesis and cell differentiation based on evolutionary large-scale chaos. *Artificial Life*, 2:79–99.

Korning P.G. (1994) Training of neural networks by means of genetic algorithm working on very long chromosomes. Technical Report, Computer Science Department, Aarhus C, Denmark.

Koza J.R. and Rice J.P. (1991) Genetic generation of both the weights and architecture for a neural network. In, *Intern. Joint Conf. on Neural Networks, Seattle 92.*

Miller G., Todd P. and Hedge S. (1989) Designing Neural Networks using Genetic Algorithm, In, *3rd Intern. Conf. on Genetic Algorithms*, D.J. Schaffer, ed., Morgan Kaufmann.

Mjolsness E., Sharp D.H. and Alpert B.K. (1989) Scaling, machine learning, and genetic neural nets. *Advances in Applied Mathematics, 10*, 137-163.

Montana D.J. and Davis L. (1989) Training feedforward neural networks using genetic algorithms. In *Proceedings of eleventh international joint conference on artificial intelligence* (pp. 762-767). San Mateo, CA: Morgan Kaufmann.

Mühlenbein H. (1990) Limitations of multi-layer perceptrons networks - steps towards genetic neural networks. *Parallel Computing*, 14:249–260.

Mühlenbein H. & Kindermann J. (1989). The dynamics of evolution and learning – Towards genetic neural networks. In R. Pfeifer, Z. Schreter, F. Fogelman-Soulie & L. Steels (Eds.), *Connectionism in perspective* (pp. 173-197). Amsterdam: Elsevier Science Publishers B.V. (North-Holland).

Nolfi S., Elman J.L. and Parisi D. (1990) *Learning and evolution in neural networks.* CRL Technical Report 9019, La Jolla, CA: University of California at San Diego.

Porto V.W. and Fogel D.B. (1990) Neural network techniques for navigation of AUVs. *Proceedings of the IEEE Symposium on Autonomous Underwater Vehicle Technology* (pp. 137-141). Washington, DC: IEEE.

Radcliffe N.J. (1990) *Genetic neural networks on MIMD computers.* Doctoral dissertation, University of Edinburgh, Edinburgh, Scotland.

Radcliffe N.J. (1991) *Genetic set recombination and its application to neural network topology optimization.* Technical report EPCC-TR-91-21, University of Edinburgh, Edinburgh, Scotland.

Rogers D. (1990) Predicting Weather Using a Genetic Memory: a Combination of Kanerva's Sparse Distributed Memory with Holland's Genetic Algorithm; *Advances in Neural Information Processing 2.*

Sutton R. (1988) Learning to Predict by the Methods of Temporal Differences, *Machine Learning,* 3:9-44.

Skinner A. and Broughton J.Q. (1995) Neural Networks in Computational Materials Science: Training Algorithms *Modelling and Simulation in Materials Science and Engineering,* 3:371—390.

Schaffer J.D., Whitley D. and Eshelman L. (1992) Combination of Genetic Algorithms and Neural Networks: The state of the art. *Combination of Genetic Algorithms and Neural Networks,* IEEE Computer Society, 1992.

Schaffer J.D., Caruana R.A. and Eshelman L.J. (1990) Using genetic search to exploit the emergent behavior of neural networks. In S. Forrest (Ed.), *Emergent computation* (pp. 244-248). Amsterdam: North Holland.

Weiland A.P. (1990) Evolving controls for unstable systems. In D.S. Touretsky, J.L. Elman, T.J Sejnowski & G.E. Hinton (Eds.) *Proceedings of the 1990 connectionist models summer school* (pp. 91-102). San Mateo, CA: Morgan Kaufmann.

Weiland A.P. (1991) Evolving neural network controllers for unstable systems. *IEEE international joint conference on neural networks* (pp. II-667 - II-673). Seattle, WA: IEEE.

Wilson S.W. (1990) Perceptron redux: Emergence of structure. In S. Forrest (Ed.), *Emergent Computation* (pp. 249-256). Amsterdam: North Holland.

Whitley D. and Hanson T. (1989) Optimizing neural networks using faster, more accurate genetic search. In J.D. Schaffer (Ed.), *Third international conference on genetic algorithms* (pp. 391-396). San Mateo, CA: Morgan Kaufmann.

Whitley D., Starkweather T. and Bogart C. (1990) Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity. *Parallel Computing.* 14:347-361.

Whitley, D., Dominic, S. & Das, R. (1991). Genetic Reinforcement Learning with Multilayered Neural Networks. *Proc. 4th International Conf. on Genetic Algorithms,* Morgan Kaufmann.

Whitley D., Dominic S., Das R. and Anderson C. (1993) Genetic Reinforcement Learning for Neurocontrol Problems. *Machine Learning* 13:259-284.

Whitley D., Gruau F. and Pyeatt L. (1995) Cellular Encoding Applied to Neurocontrol. In, *5th Intern. Conf. on Genetic Algorithms*, L. Eshelman, ed., Morgan Kaufmann.