To appear in *Proceedings of the 2003 IEEE Congress on Evolutionary Computation (CEC-2003)*

Evolving Adaptive Neural Networks with and without Adaptive Synapses

Kenneth O. Stanley
Dept. of Computer Sciences
The University of Texas at Austin
Austin, TX 78705
kstanley@cs.utexas.edu

Bobby D. Bryant
Dept. of Computer Sciences
The University of Texas at Austin
Austin, TX 78705
bdbryant@cs.utexas.edu

Risto Miikkulainen
Dept. of Computer Sciences
The University of Texas at Austin
Austin, TX 78705
risto@cs.utexas.edu

Abstract- A potentially powerful application of evolutionary computation (EC) is to evolve neural networks for automated control tasks. However, in such tasks environments can be unpredictable and fixed control policies may fail when conditions suddenly change. Thus, there is a need to evolve neural networks that can *adapt*, i.e. change their control policy dynamically as conditions change. In this paper, we examine two methods for evolving neural networks with dynamic policies. The first method evolves recurrent neural networks with fixed connection weights, relying on internal state changes to lead to changes in behavior. The second method evolves local rules that govern connection weight changes. The surprising experimental result is that the former method can be more effective than evolving networks with dynamic weights, calling into question the intuitive notion that networks with dynamic synapses are necessary for evolving solutions to adaptive tasks.

1 Introduction

In evolutionary computation (EC), evolved solutions are usually static: Their parameters do not change during performance. However, in many important control problems, the environment may change suddenly or gradually, and to maintain sufficient performance, the controller needs to adapt to it online. For example, a robot may lose a sensor or an airplane may lose an engine and there may not be any opportunity to re-evolve controllers for such unexpected circumstances. Adaptation is necessary also to make the controllers general. For example, a controller evolved for driving a car should work for any car, even if the dimensions and mechanics are slightly different from its training models.

Natural organisms are constantly faced with unforeseen circumstances and generally adapt to them very well. They can do it because their nervous systems are plastic, i.e. not fixed at birth. Thus, one way to achieve adaptive solutions is to evolve neural networks with *plastic synapses*, i.e. adaptive networks. Evolution can discover parameterizations that lead to robust adaptability.

Several researchers have evolved neural systems that learn. Nolfi and Parisi used backpropagation inside evolved networks to make predictions about future states (Nolfi et al. 1990, 1994) and also for a network to train itself (Nolfi and Parisi 1993, 1995). Chalmers (1990) evolved a global learning rule that turned out to be similar to the delta rule. McQuesten and Miikkulainen (1997) showed that parent networks can teach their offspring using backpropagation. These experiments showed that *general* learning mechanisms can improve performance of evolved neural networks.

However, the goal of evolving adaptive neural networks is different from evolving networks that simply utilize learning. The goal is to find *specific* learning mechanisms that are optimized to adapt to new or changed problems that can arise in the domain. In a significant demonstration of the power of this approach, Floreano and Urzelai (2000) evolved neural networks with local synaptic plasticity parameters and compared them to fixed-weight networks in a two-step task. The networks were evolved to turn on a light, and then move to a gray square. The local learning rules, they found, helped networks quickly alter their functionality, facilitating a policy transition from one task to another.

While Floreano and Urzelai's experiment demonstrated that evolving local learning rules can be useful, a further question is whether such adaptation is actually *necessary* in tasks that *require* adaptation. In such tasks the correct policy depends on an aspect of the environment that varies randomly from one trial to the next and is not immediately observable but must be discovered through exploration. Since synaptic plasticity allows natural organisms to adapt to novel situations, we would expect plastic neurons to allow artificial neural networks to adapt similarly. Yet although synaptic plasticity facilitates such adaptation, recurrent networks with fixed weights can also respond differently in changing environmental conditions. Thus, two important questions are, (1) are plastic synapses necessary to adapt to changing environments, and (2) does adding local learning rules support the network's ability to adapt when necessary? To answer these questions, we evolved fixed-weight networks and adaptive neural networks in a food foraging domain designed to require a policy change during the network's lifetime. We used the NeuroEvolution of Augmenting Topologies (NEAT) method, augmented with a facility for evolving local Hebbian learning rules for specific connections in the network. Thus, the connection weights could change over the network's lifetime according to different evolved rules at each connection.

The results indeed confirmed that networks with evolved plasticity were able to adapt to change in the environment. However, the experiments also yielded a surprising result: Recurrent networks with fixed connection weights could also solve the task. Moreover, the fixed-weight networks did so faster and more reliably than the adaptive networks. Evolution found an alternative, clever way of implementing adaptation in this problem: The recurrent network changes its internal state to change the behavior policy of the network. This result implies that adaptation does not always require synaptic plasticity, although evolution can utilize plasticity if it is available. Further, because local learning parameters expand the genetic search space, which can make finding a solution more difficult, local learning may not always be the right approach for adaptive tasks.

We begin the paper by describing the NEAT method for evolving neural networks, and how NEAT was augmented to evolve learning parameters in addition to connection weights. We then describe the food foraging domain used in the experiments, and present the results.

2 Adaptive NeuroEvolution of Augmenting Topologies

The parameter space being searched in the space of adaptive networks is much larger than for static networks. In addition to connection weights, several learning parameters must be evolved for each learning rule at each connection. Thus, it is important to minimize the number of connections optimized by evolution. In addition, the topology of the network has to be optimized. It is necessary to have connections between the *right* nodes, so that the connections can be strengthened or weakened to change the relationship between the computational concepts they represent.

In order to discover minimal effective adaptive topologies, we used the NeuroEvolution of Augmenting Topologies (NEAT) method (Stanley and Miikkulainen 2002d). NEAT combines the usual search for appropriate network weights with complexification of the network structure. This approach is highly effective: NEAT outperforms other neuroevolution (NE) methods, e.g. on the benchmark double pole balancing task (Stanley and Miikkulainen 2002c,d). In addition, because NEAT starts with simple networks and expands the search space only when beneficial, it is able to find significantly more complex controllers than fixed-topology evolution, as demonstrated in a robotic strategy-learning domain (Stanley and Miikkulainen 2002a,b). These properties make NEAT an attractive method for evolving adaptive neural networks.

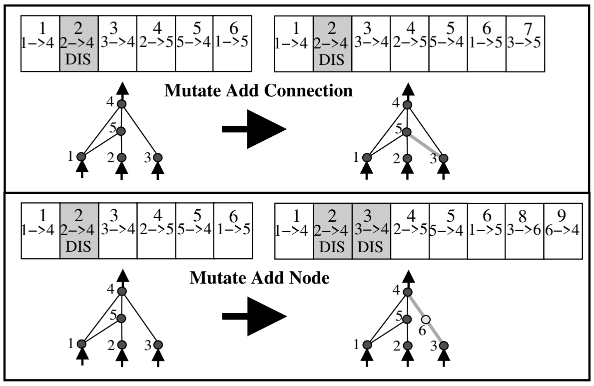


Figure 1: **The two types of structural mutation in NEAT.** Both types are illustrated with the genes above their phenotypes. The top number in each gene is the *innovation number* of that gene. The bottom two numbers denote the two nodes connected by that gene. The weight of the connection, also encoded in the gene, is not shown. The symbol *DIS* means that the gene is disabled. Assuming the depicted mutations occurred one after the other, the genes would be assigned increasing innovation numbers as the figure illustrates.

In this section, we briefly review the original NEAT method, and further describe how it can be extended to efficiently evolve parameters for learning functions at each synapse.[1]

2.1 Genetic Encoding with Historical Markings

Evolving network structure requires a flexible genetic encoding. Each genome in NEAT includes a list of *connection genes*, each of which refers to two *node genes* being connected. Each connection gene specifies the in-node, the out-node, the weight of the connection, whether or not the connection gene is expressed (an enable bit), and an *innovation number*, which allows finding corresponding genes during crossover.

Mutation in NEAT can change both connection weights and network structures. Connection weights mutate as in any NE system, with each connection either perturbed or not. Structural mutations, which allow complexity to increase, either add a new connection or new node to the network (figure 1). Through mutation, genomes of varying sizes are created, sometimes with completely different connections specified at the same positions.

In order to perform crossover, the system must be able to tell which genes match up between *any* individuals in the population. NEAT keeps track of which genes line up with which by keeping track of the historical origin of ev-

[1] A more comprehensive description of the NEAT method is given by Stanley and Miikkulainen (2002d).

ery gene. Whenever a new gene appears (through structural mutation), a *global innovation number* is incremented and assigned to that gene. The innovation numbers thus represent a chronology of every gene in the system (figure 1). Whenever these genomes crossover, innovation numbers on inherited genes are preserved. Thus, the historical origin of every gene in the system is known throughout evolution.

Through innovation numbers, the system now knows exactly which genes match up with which. Genes that do not match are either *disjoint* or *excess*, depending on whether they occur within or outside the range of the other parent's innovation numbers. When crossing over, the genes in both genomes with the same innovation numbers are lined up. Genes that do not match are inherited from the more fit parent, or if they are equally fit, from both parents randomly.

Historical markings allow NEAT to perform crossover without the need for expensive topological analysis. Genomes of different organizations and sizes stay compatible throughout evolution, and the problem of matching different topologies (Radcliffe 1993) is essentially avoided.

2.2 Protecting Innovation through Speciation

Adding new structure to a network usually initially reduces fitness. However, NEAT speciates the population, so that individuals compete primarily within their own niches instead of with the population at large. This way, topological innovations are protected and have time to optimize their structure before they have to compete with other niches in the population.

Historical markings make it possible for the system to divide the population into species based on topological similarity. The distance δ between two network encodings is a simple linear combination of the number of excess (E) and disjoint (D) genes, as well as the average weight differences of matching genes ($\overline{W}$):

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot \overline{W}. \tag{1}$$

The coefficients c_1, c_2, and c_3 adjust the importance of the three factors, and the factor N, the number of genes in the larger genome, normalizes for genome size. Genomes are tested one at a time; if a genome's distance to a randomly chosen member of the species is less than δ_t, a compatibility threshold, it is placed into this species. Each genome is placed into the first species where this condition is satisfied, so that no genome is in more than one species.

The reproduction mechanism for NEAT is *explicit fitness sharing* (Goldberg and Richardson 1987), where organisms in the same species must share the fitness of their niche, preventing any one species from taking over the population.

2.3 Minimizing Dimensionality

Unlike other systems that evolve network topologies and weights (Gruau et al. 1996; Yao 1999) NEAT begins with a uniform population of simple networks with no hidden nodes. New structure is introduced incrementally as structural mutations occur, and only those structures survive that are found to be useful through fitness evaluations. This way, NEAT searches through a minimal number of weight dimensions.

2.4 Evolving Local Learning Rules

Each connection in an adapting network follows a rule that governs how its weight changes. We implemented local learning rules based on those used by Floreano and Urzelai (2000). Floreano and Urzelai chose their rules based on synaptic mechanisms observed in mammals (Willshaw and Dayan 1990). The space of possible rules included a *plain Hebbian rule*, which strengthens the connection proportionally to correlated activation, and rules for weakening the connection when activations do not correlate.

We streamlined Floreano and Urzelai's rules to make them more efficient and to fit NEAT's genetic encoding more naturally. Instead of dividing Hebbian learning into separate rules, we evolved a single general learning rule for both excitatory and inhibitory connections that combines the properties of Floreano and Urzelai's rules. That way, only two parameters are needed to express a rule, keeping the search space to a minimum. Let x and y be the activities of the incoming and outgoing neurons, respectively, and W be the highest weight magnitude in the network. If the connection is excitatory, the change in weight magnitude Δw can be expressed as

$$\Delta w = \eta_1 (W - w) x y + \eta_2 W x (y - 1.0) , \tag{2}$$

where η_1 is the Hebbian learning rate and η_2 is the decay rate, which controls how fast the connection weakens when the presynaptic node does not affect the postsynaptic node. Inhibitory connections adapt as

$$\Delta w = -\eta_1 (W - w) x y + \eta_2 (W - w) x (1.0 - y) . \tag{3}$$

The first term in equation 3 is negative because correlated activation implies that the connection does not have an inhibitory effect. The second term strengthens the connection when the input is high and the output is low, increasing the contribution of the inhibitory connection. Equations 2 and 3 were validated through extensive preliminary experimentation.

To implement equations 2 and 3, every connection in an adaptive NEAT neural network has a local learning rule parameterized by η_1 and η_2, in addition to its evolved weight.

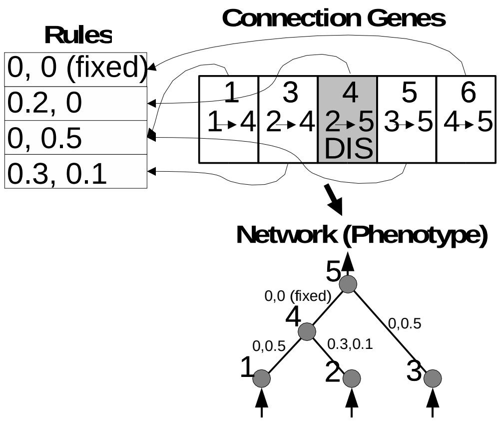


Figure 2: **Encoding Local Adaptation Rules.** The connection genes are depicted as in figure 1, with gene 4 disabled. Each connection gene points to one of the four local learning rules. The corresponding connection in the phenotype receives the learning parameters pointed to by its respective gene. This way, fewer parameters need to be optimized, and rules can be reused. The connections between nodes 1 and 4 and between 3 and 5 both use the same rule.

Yet if every connection gene expressed its own local learning parameters, the dimensionality of the parameter space would multiply by a factor of 3, reducing the chance of finding a solution. Moreover, it is likely that many connections will utilize the same rule. It should not be necessary to rediscover such a rule for every connection that uses it.

Thus, we augmented the NEAT genetic encoding so that the rules could be reused by more than one connection gene. Instead of having all of the adaptation parameters for each connection and node expressed in every gene, each genome has a single *rule set*. This set consists of a finite number of rules, in our implementation one "fixed weight" rule and three evolved rules, each containing the adaptation parameters η_1 and η_2. Each gene points to one rule in the rule set. Thus, when a genome is translated into a network, the connections and nodes receive the parameters of the rule to which their genes point (Figure 2). These pointers can change through mutation. This system accomplishes 3 objectives: (1) The number of learning parameters in a genome *does not* increase as the genome grows; (2) the same rules can be *reused* by many genes; and (3) the adaptation rules can be optimized separately from the connection genes. Thus, using the rule set in NEAT can be seen as an efficient alternative to evolving separate rules for every gene.

The question, then, is whether evolution can take advantage of adaptive synapses when necessary. The next section describes a domain designed to answer this question.

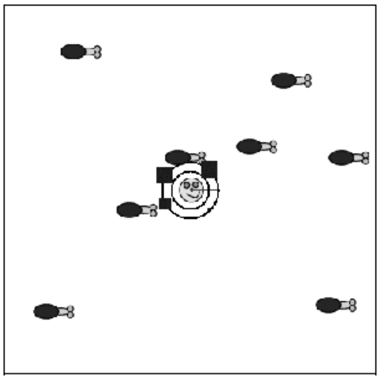


Figure 3: **The Dangerous Food Foraging Domain.** The robot begins in the center of the field. The concentric circles around the robot represent one ring of sensors for type A items and one ring for type B. Eight items are dispersed randomly throughout the field. In this case, the items are type B. However, they may or may not be poisonous. The robot can only find out if the items are poisonous by consuming them, after which is must stop foraging if it senses pain. This domain requires adaptation because a fixed network cannot change its policy in midcourse.

3 The Dangerous Foraging Domain

In order to analyze the evolution of adaptation, a domain is needed in which adaptation is necessary and where success can be readily measured. Such a domain can be constructed by making the optimal policy depend upon a hidden property of the world that can only be discovered through exploration. That way, once the hidden property is uncovered, the policy must adapt accordingly. A fixed policy, e.g. a neural network with only fixed-weight feedforward connections, cannot change its policy, and thus cannot succeed in such a domain. Networks must evolve to adapt to properties native to the particular problem.

One such problem occurs in natural foraging. When an animal enters a new geographical area, some food may first appear edible yet turn out poisonous. The animal must be able to change its policy and stop gathering such food. We implemented a food foraging domain based on this dangerous natural scenario. Some of the food is poisonous and some is edible. However, it is not possible tell which is which without trying the food first. A simulated robot begins in the center of a field with one of two types of item, type A, or type B, spread randomly throughout (figure 3). The items are either all poison, or all food, and the robot must consume at least one item to find out whether it should continue foraging. After consuming an item, the neural network receives a brief pain or pleasure signal. The correct policy thereafter depends upon this signal.

Because the correct policy, whether to forage or not, cannot be fixed at the start, the network must be able to adapt.

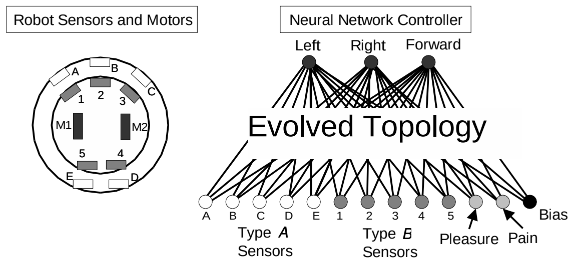


Figure 4: **The Robot and its Controller Network.** Five type A sensors and five type B sensors detect the presence of objects around the robot. The pleasure or pain sensors activate for 20 time steps (out of a total of 750 in each trial) if the robot consumes food or poison. These signals give the robot a chance to change its behavior accordingly. The three motor outputs are mapped to forces that control the left and right wheels. Evolution must discover a network that can change the control policy of the robot when it encounters poison.

The challenge of constructing such a network is significant because it must evolve policies for foraging items of both type A and type B, but also be capable of abandoning either policy if poison is encountered.

The simulated robots are similar to Kheperas (Mondada et al. 1993). Each has two wheels controlled by separate motors. Five rangefinder sensors can sense type A items and five can sense type B (figure 4). Finally, each robot has a pleasure sensor, which activates when it consumes food, and a pain sensor, which activates when it consumes poison. The pleasure/pain sensors are used for adaptation.

4 Experiments

The experiments are designed to test the hypothesis that adaptive synapses are necessary for adaptation. Thus, we test (1) whether fixed-weight networks with recurrent connections can solve the task and (2) whether allowing synapses to adapt enhances evolution's capacity to discover adaptive solutions.

Accordingly, we ran five 350-generation runs with fixed connection weights and five 500-generation runs with plastic synapses (the former runs took fewer generations to converge). All runs were free to utilize recurrent connections but only the adaptive runs could utilize adaptive synapses. Each run took approximately 2 days to complete on a 1.8 Ghz Pentium 4 processor. The NEAT algorithm itself took less than 1% of this computation: The rest of the time was spent evaluating networks in the foraging task.

4.1 Experimental Setup

In each run, the population consisted of 500 NEAT networks. Each network was evaluated in eight separate trials: Two trials with edible type A items, two trials with edible type B items, two trials with poisonous type A items, and two trials with poisonous type B items. Networks were reset to their initial state before each trial, that is, internal activations were flushed to zero and synapses were reset to the initial weights defined in the genome.

Fitness was evaluated by rewarding networks for consuming food and penalizing them for consuming poison. Since there were 8 items in each trial, and 4 trials consisted of poison, the maximum number of poison consumed is 32. Thus, in order to ensure positive fitness values, the fitness function f was defined,

$$f = 32 + e - p \,, \tag{4}$$

where e is the number of edible items consumed and p is the number of poisonous items. The maximum possible fitness is 64. However, in general the highest fitness that can be consistently attained is 60 because the only way to know when poison is present is by testing at least one item in the field. Thus, since there are four poison trials, the best networks need to consume four poisons to properly test for poison in each trial.

Because the fitness evaluation has a degree of randomness, i.e. the placement of food is random in each trial, the fitness function is noisy. Thus, simply reaching a fitness of 60 is not sufficient to indicate a solution. Rather, a solution has been reached when the population champion consistently reaches a fitness of 60 for several generations in a row.

4.2 Parameter Settings

The coefficients for measuring compatibility were $c_1 = 1.0$, $c_2 = 1.0$, and $c_3 = 2.0$. The initial compatibility distance was $\delta_t = 6.0$. However, because population dynamics can be unpredictable over hundreds of generations, we assigned a target of 20 species. If the number of species grew above 20, δ_t was increased by 0.3 to reduce the number of species. Conversely, if the number of species fell below 20, δ_t was decreased by 0.3 to increase the number of species. In order to prevent stagnation, the lowest performing species over 130 generations old was not allowed to reproduce. The champion of each species with more than five networks was copied into the next generation unchanged. The interspecies mating rate was 0.01. The probability of adding a new node was 0.005 and the probability of a new link mutation was 0.05. These parameter values were found experimentally but they do follow intuitively meaningful rules: Links need to be added significantly more often than nodes, and an average weight difference of 0.5 is about as significant as one disjoint or excess gene. Performance is robust to moderate variations in these values: The dynamic compatibility distance measure caused speciation to remain stable.

In the adaptive runs, genomes contained four rules with values for η_1 and η_2. The first rule was fixed at zero to allow genes to encode fixed connection weights that do not adapt. The probability of mutating the parameters of a rule was 0.3, in which case each parameter mutated with a probability of 0.2 by adding uniform random noise between 0 and 0.5. A connection gene had a 0.05 chance of being pointed to a different rule.

The learning rules affect how compatible network genomes are. Thus they were factored into network compatibility calculations. The average difference in learning rule parameter values was added to $\overline{W}$ in calculating the compatibility of two genes. That way, speciation took into account learning rule differences in addition to topology and weight differences.

5 Results

5.1 Evolving Fixed-Weight Neural Networks

Figure 5 shows that fixed-weight networks were able to solve the task. In fact, all five runs could consistently score 60 or above before 350 generations.[2] The best run found a consistent solution by the 250th generation.

5.2 Evolving Adaptive Neural Networks

NEAT was able to evolve networks with local learning rules that solved the task. However, solutions with adaptive synapses were more difficult to find. Figure 6 shows how fitness increased over generations in both the average and best runs of adaptive evolution. Three of the five runs converged to consistently scoring 60 or above, whereas the other two runs never found a consistent solution. Thus, the variance in fitness was higher for adaptive networks than for fixed-weight networks. The best run was able to find a consistent solution by the 350th generation, 100 generations later than fixed-weight evolution! Nevertheless, since several runs did find solutions, these results show that local Hebbian learning can be utilized to encode dynamic policies.

The unexpected conclusion is that for this task, evolution of fixed-weight recurrent networks is not only sufficient to solve the task, but more efficient than evolving adaptive networks. How does this result come about?

5.3 Typical Solution Networks

Let us analyze solution networks from each type of evolution in order to understand how each kind of network represents a policy that can change over time, and why fixed-weight networks evolved faster and more reliably.

Figure 7 depicts typical solutions from each type of evolution. It turns out that the fixed-weight solution uses a simpler mechanism than the adaptive solution. Most of the structure in the fixed-weight network, including recurrent connections on hidden nodes, is used to stabilize food gathering trajectories, rather than to modulate behavior depending on pain or pleasure. In fact, the network can still reliably perform the task even if all its hidden nodes are ablated, although it takes longer because its actions are less accurate.

The key components of the fixed-weight solution are the self-recurrent connections on the output nodes. The strong excitatory self-recurrent connection on the *left turn* output (identified in figure 4) keeps the node active even when it has low input. The only thing that can stop the self-activation cycle is a strong inhibitory signal directly from the pain sensor, in which case the left turn output node is temporarily disabled. At the same time, the *right turn* output turns off unless food is directly in front of the robot. Thus, when the left turn output is temporarily muted, the right turn output will cause the robot to spin until it is facing away from food. At this point both turn outputs will be off, and the robot will dash forward through the open space to a wall, thus avoiding any further food gathering. Thus, recurrent connections *are* used to represent a dynamic policy. This clever solution demonstrates the power of evolution in finding effective novel solutions.

In contrast, the adaptive solution uses its hidden nodes as part of its policy-changing method. If hidden nodes are ablated, the network can no longer perform the task. 22% of the connections, i.e. 16 connections, diverge in opposite directions depending on whether or not the robot discovers it is in a food or poison trial. Of those, 8 connections diverge in *both* type A and type B trials. In other words, an *abstraction* of the food vs. poison distinction evolved that is independent from what the objects look like. Thus, the solution is holistic in that the entire network contributes to the task in every trial. Instead of driving into a wall after consuming poison like the recurrent solution, the adaptive network spins in place.

Other recurrent and adaptive solutions followed similar patterns; recurrent solutions tended to rely exclusively on a "trick" using recurrent connections on output nodes, while adaptive solutions tended to make complex internal network changes. These results explain why recurrent solutions were found more easily. These networks only need to find a particular combination of recurrent connections on outputs, and then use hidden nodes to refine trajectory control. Adaptive networks, on the other hand, tend to discover complex holistic solutions that genuinely change network functionality.

[2] The reason networks sometimes scored above 60 is that some species evolved a small probability of doing nothing during a trial. Although this behavior sometimes led to a significant drop in fitness when a food trial was missed, in some cases the networks got lucky and skipped a poison trial.

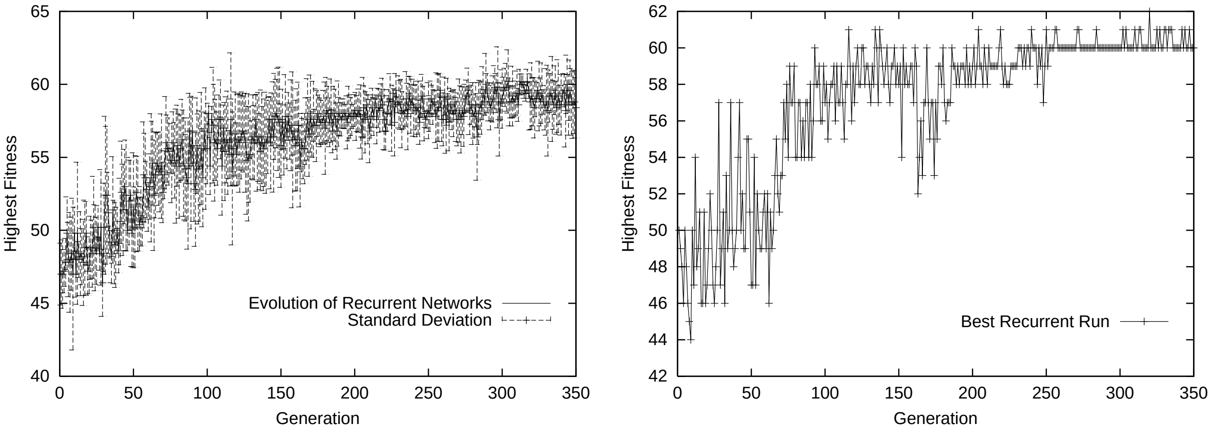


Figure 5: **Average Highest Fitness and Best Run of Networks with Fixed-Weight Neural Networks.** The graphs depict the highest average fitness and the highest fitness of the best run over generations of fixed-weight recurrent evolution. All five runs discovered solutions within 350 generations, demonstrating that dynamic synapses are not necessary for adaptation in this task.

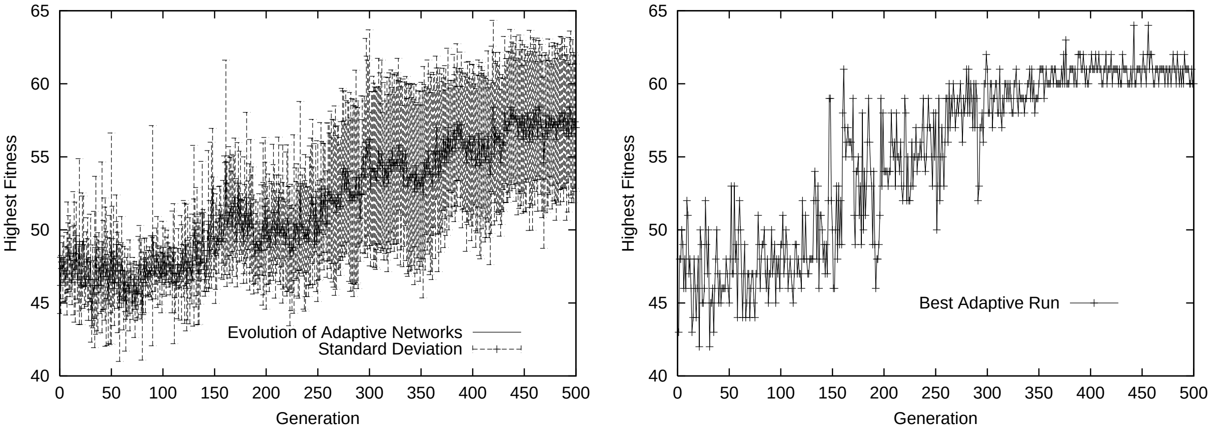


Figure 6: **Average Highest Fitness and Best Run of Adapting NEAT.** The graphs show how the highest fitness increased in adaptive evolution both on average and in the best run. Three of five runs found a consistent solution. Thus, the task can be solved with adaptive synapses, although not as reliably as with fixed-weight synapses.

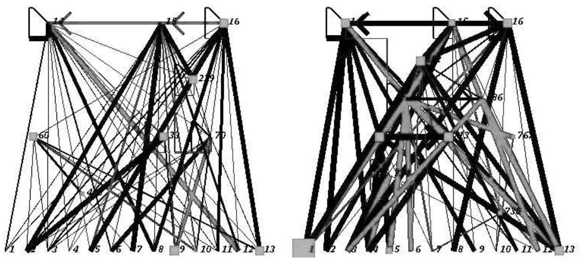
6 Discussion and Future Work

Although evolution of adaptive networks was successful in this task, the surprising result is that fixed-weight recurrent networks were able to solve it with fewer evaluations. Recurrency alone was successful in this experiment because an unanticipated yet relatively simple solution existed in the space of fixed-weight recurrent networks. This result suggests that if such a solution exists, finding it can be easier than finding an adaptive solution. The general conclusion is then that justification is required before broadening the search to include learning parameters.

Nevertheless, local learning rules may provide an advantage in some domains. The holistic nature of evolved adaptive networks suggests that such networks may be superior to fixed-weight recurrent networks on tasks that require holistic solutions. A major problem for future research is to characterize what kinds of tasks do.

Why did the adaptive networks, which were also able to use recurrent connections, not evolve the same style of solutions as the fixed-weight networks? As connection weights became dynamic early in evolution, trivial solutions based on recurrent output nodes were no longer feasible because the network weight configuration was not reliable. Thus, evolution was forced to utilize the dynamic synapses in order to master the task, leading to more holistic solutions.

An important question for future research is whether fixed-weight recurrent networks can scale up to more difficult tasks, or whether there exist some tasks for which dynamic synapses and holistic solutions are necessary. Characterizing those situations where dynamic synapses provide an advantage will contribute to our general understanding of adaptation and help to explain its use in nature.



(a) Fixed−Weight Recurrent **(b) Adaptive**

Figure 7: **Solution Network Examples.** Typical solutions are depicted for adaptive evolution and fixed-weight recurrent evolution. Nodes are shown as squares beside their node numbers, and line thickness represents the strength of connections. Dark lines represent excitatory connections, and light lines are inhibitory connections. Loops at nodes represent self-recurrency. (a) The fixed-weight network solves the task using recurrent connections on its outputs. (b) The adaptive solution is holistic, utilizing plastic synapses throughout the network.

7 Conclusion

Both fixed-weight networks and networks with dynamic synapses were evolved in a dangerous food foraging task. The only way to succeed in the task is to be able to switch off the foraging behavior in the middle of a trial. Although adaptive networks seem well-suited for a such a task, in fact the fixed-weight networks evolved solutions more quickly and more reliably. The fixed-weight solutions exploited a clever strategy of switching their internal state, represented by recurrent connections, while adaptive evolution found more complex holistic solutions. The conclusion is that recurrency alone may be a sufficient method of adaptation in many tasks, and because of its smaller search space, may be easier to evolve than solutions with adaptive synapses.

Acknowledgments

This research was supported in part by the National Science Foundation under grant IIS-0083776 and by the Texas Higher Education Coordinating Board under grant ARP-003658-476-2001.

Bibliography

Chalmers, D. J. (1990). The evolution of learning: An experiment in genetic connectionism. In Touretzky, D. S., Elman, J. L., Sejnowski, T. J., and Hinton, G. E., editors, *Connectionist Models: Proceedings of the 1990 Summer School*, 81–90. San Francisco, CA: Morgan Kaufmann.

Floreano, D., and Urzelai, J. (2000). Evolutionary robots with on-line self-organization and behavioral fitness. *Neural Networks*, 13:431–4434.

Goldberg, D. E., and Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In Grefenstette, J. J., editor, *Proceedings of the Second International Conference on Genetic Algorithms*, 148–154. San Francisco, CA: Morgan Kaufmann.

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

McQuesten, P., and Miikkulainen, R. (1997). Culling and teaching in neuro-evolution. In Bäck, T., editor, *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA-97, East Lansing, MI)*, 760–767. San Francisco, CA: Morgan Kaufmann.

Mondada, F., Franzi, E., and Ienne, P. (1993). Mobile robot miniaturization: A tool for investigation in control algorithms. In *Proceedings of the Third International Symposium on Experimental Robotics*, 501–513.

Nolfi, S., Elman, J. L., and Parisi, D. (1990). Learning and evolution in neural networks. Technical Report 9019, Center for Research in Language, University of California, San Diego.

Nolfi, S., Elman, J. L., and Parisi, D. (1994). Learning and evolution in neural networks. *Adaptive Behavior*, 2:5–28.

Nolfi, S., and Parisi, D. (1993). Auto-teaching: Networks that develop their own teaching input. In Deneubourg, J. L., Bersini, H., Goss, S., Nicolis, G., and Dagonnier, R., editors, *Proceedings of the Second European Conference on Artificial Life*, 845–862.

Nolfi, S., and Parisi, D. (1995). Learning to adapt to changing environments in evolving neural networks. Technical Report 95-15, Institute of Psychology, National Research Council, Rome, Italy.

Radcliffe, N. J. (1993). Genetic set recombination and its application to neural network topology optimization. *Neural computing and applications*, 1(1):67–90.

Stanley, K. O., and Miikkulainen, R. (2002a). Competitive coevolution through evolutionary complexification. Technical Report AI2002-298, Department of Computer Sciences, The University of Texas at Austin.

Stanley, K. O., and Miikkulainen, R. (2002b). Continual coevolution through complexification. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*. San Francisco, CA: Morgan Kaufmann.

Stanley, K. O., and Miikkulainen, R. (2002c). Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*. San Francisco, CA: Morgan Kaufmann.

Stanley, K. O., and Miikkulainen, R. (2002d). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127.

Willshaw, D., and Dayan, P. (1990). Optimal plasticity from matrix memories: What goes up must come down. *Neural Computation*, 2:85–93.

Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447.