The Evolution of Learning: An Experiment in Genetic Connectionism

David J. Chalmers

Center for Research on Concepts and Cognition
Indiana University
Bloomington, Indiana 47408.
E-mail: dave@cogsci.indiana.edu

Abstract

This paper explores how an evolutionary process can produce systems that learn. A general framework for the evolution of learning is outlined, and is applied to the task of evolving mechanisms suitable for supervised learning in single-layer neural networks. Dynamic properties of a network's information-processing capacity are encoded genetically, and these properties are subjected to selective pressure based on their success in producing adaptive behavior in diverse environments. As a result of selection and genetic recombination, various successful learning mechanisms evolve, including the well-known delta rule. The effect of environmental diversity on the evolution of learning is investigated, and the role of different kinds of emergent phenomena in genetic and connectionist systems is discussed.

In D. S. Touretzky, J. L. Elman, T. J. Sejnowski & G. E. Hinton (eds.), *Proceedings of the 1990 Connectionist Models Summer School.* San Mateo, CA: Morgan Kaufmann.

1 Introduction

Evolution and learning are perhaps the two most fundamental processes of adaptation, and their relationship is very complex. This paper explores one aspect of this relationship: how, via the process of evolution, the process of learning might evolve. Taking this approach, we can view evolution as a kind of second-order adaptation: it is a process that produces individual systems that are not immediately adapted to their environment, but that have the ability to adapt themselves to many environments by the first-order adaptive process of learning. Here the learning mechanisms themselves are the object of evolution. Using a combination of methods drawn from connectionism and genetic search, we will start with a population of individuals with no ability to learn, and attempt to evolve useful learning techniques.

The details of the experiments performed here start in the second section. The remainder of this introduction is devoted to a somewhat philosophical discussion of the role of evolutionary computation in the modeling of cognitive systems, and in particular the importance of the phenomenon of emergence. A brief outline of related work combining connectionist and genetic methods is also given. Readers who wish to skip this background material may proceed directly to the second section.

1.1 Evolution, Emergence and Computation

In recent years, the evolutionary approach to the computational modeling of cognitive systems has gained prominence, following the work of Holland and his colleagues on genetic algorithms (Holland 1975; Goldberg 1989), a class of methods of search directly inspired by biological systems. The attractions of this approach are many. To the biologist, ethologist or psychologist, evolutionary computation offers insight into the mechanisms that gave rise to adaptations present in existing living systems. To the computer scientist or the engineer, genetic search can yield a powerful method of problem solving. If we accept the often-repeated slogan, "Nature is smarter than we are", then harnessing the problem-solving methods that nature uses makes sound practical sense. Finally, to a cognitive scientist, evolutionary methods offer a paradigm of *emergence* that seems to have much in common with the kind of emergence found in contemporary connectionist systems. In both kinds of systems, complex high-level behavior is produced as a result of combining simple low-level computational mechanisms in simple ways.

The kind of emergence found in genetically-based systems differs, however, from that found in connectionist systems. Connectionist systems support *synchronic emergence*, or emergence over levels: at a given time, a host of low-level computations are taking place, which when looked at on another level can be interpreted as complex high-level functioning. By contrast, genetically-based systems support *diachronic emergence*, or emergence over time: primitive computational systems gradually evolve towards greater complexity.

To the cognitive scientist, while there is a good deal of interest in the *process* of evolution, the greatest interest lies in the *product*. One wishes that eventually, as a product of the appropriate sort of genetic search, we will be able to produce plausibly functioning cognitive systems. This is where the distinction between synchronic and diachronic emergence becomes important. The diachronic emergence of the evolutionary process gives no guarantee that the systems that result from the process will manifest the kind of emergence-over-levels that is so important in connectionist systems. For example, classifier systems, the most prominent application of genetically-based computation (Holland 1986), typically give rise via genetic search to a kind of production system, a paradigm of symbolic computation. To someone — a connectionist, for example — who believes that emergence-over-levels is fundamental to most cognitive phenomena, this is somewhat unsatisfying. This is not to say that classifier systems are uninteresting — they have yielded powerful insights into the process of learning, for instance (see e.g. Holland, Holyoak, Nisbett and Thagard 1986) — but it would be very interesting to see whether evolutionary methods could be used to develop systems that might be classed as emergent in their own right.

The road to achieving synchronic emergence through evolutionary methods seems clear: loosen the connection between genotype and phenotype[1]. In classifier systems (and in most other current applications of genetic algorithms) the connection between these is very direct. An element of the genome codes for a feature-detector or an action in the phenotype in a direct, symbolic fashion. This contrasts strongly with the human case, in which elements of the genome appear to determine phenotypic functioning in only a very indirect way, coding for low-level mechanisms that produce human behavior via "action at a distance". When the genotype encodes high-level features directly and symbolically, there is no room for synchronic emergence. Emergence-over-levels can only take place when levels are distinct. The key to achieving full emergence with evolutionary systems seems to lie in the ability to allow

1. The genotype is the collection of genetic information passed on between generations: in a genetic algorithm, this is typically a string of bits. The phenotype is the behavioral expression of the genotype, an entity that interacts with an environment and is subject to selection by differential fitness. In the human case, genotype=DNA, phenotype=person.

phenotypic characteristics to emerge indirectly from genetic information.

There is another motivation for indirect mappings from genotype to phenotype: open-ended search. Current genetically-based methods have been criticized for not allowing a sufficiently "open-ended" space of possibilities for evolution. It is a feature of current genetic search methods that a genotypic space is precisely specified in advance, and search cannot go outside this space. When this is coupled with a direct genotype-to-phenotype mapping, it translates directly into a strongly-delineated phenotypic space whose properties are well-understood in advance. The role of genetic search is reduced to that of optimizing over such a well-understood space. However, if the relationship between genotype and phenotype is indirect and emergent, phenotypic space need not be so strongly constrained. To be more precise, in such cases phenotypic space will be indirectly constrained by the genotypic space, but synchronic emergence guarantees that high-level phenotypic characteristics need not be limited in advance by the imagination of the designer.

There are at least two different ways to achieve such an emergent relationship between genotype and phenotype. The first is to make the genotype code for *subsymbolic* computation in the phenotype: it might, for example, determine the low-level computations of a connectionist system, without explicitly constraining high-level behavior. Secondly, a genotype might code for *developmental* processes in the phenotype. On this approach, the phenotype's final form would not be specified, but instead would result from a complex pattern of developmental processes over its lifetime. In this paper, we will use both kinds of emergent genotype-phenotype relationships simultaneously: the genotype will specify the dynamics of a developing system that interacts with an environment, and these dynamics will control low-level properties of a connectionist system. In this way, we will see whether adaptation at the level of evolution can give rise to adaptation at the level of the individual: specifically, whether we can produce, through evolution, the capacity for a connectionist system to learn.

1.2 Genetic Connectionism

The combination of genetic search with connectionist computation is already popular. The power of genetic coding at the subsymbolic level is well-recognized. A key motivation behind this "genetic connectionism" stems from the fact that due to the synchronic emergence of connectionist systems, it is difficult to know in advance precisely which low-level computations are appropriate for a specific high-level behavior. In view of this difficulty, it makes sense to use genetic methods to search for an appropriate low-level computational form. Rather than construct the right kind of system or architecture by hand, why not let natural selection do the work?

Genetic connectionism to date has usually taken one of two forms. Genetic information has been used to encode the connection strengths in a neural network (e.g. Belew, McInerney & Schraudolph, 1990; Wilson 1990), or to encode the overall topology of a network (Miller, Todd & Hegde 1989; Whitley, Starkweather & Bogart 1989). These methods have yielded interesting results on the optimization of network design. They have not, however, led to qualitatively new kinds of connectionist processes. The reason for this lies with the fact that the phenotypic spaces are qualitatively well-understood in advance (although, as pointed out above, the relationship between individual phenotypes and genotypes may not be). Genetic methods provide a powerful method of searching these spaces, but are unlikely to come up with surprising results. Soon we will look at an application where the nature of the phenotypic space is less clear in advance.

Much interesting recent work has focused on the relationship between evolution and learning. It has become apparent that genetic connectionism is an ideal modeling tool in this endeavor: genetic algorithms providing an elegant model of evolution, and connectionism providing simple but powerful learning mechanisms. Belew, McInerney and Schraudolph (1990), following up suggestions by Maynard Smith (1987) and Hinton and Nowlan (1987), have demonstrated the complementary nature of the two phenomena: the presence of learning makes evolution much easier (all evolution has to do is find an appropriate initial state of a system, from which learning can do the rest); and evolutionary methods can significantly speed up the learning process. Nolfi, Elman and Parisi (1990) have similarly investigated the ways in which the presence of learning might facilitate the process of evolution.

These studies have presumed a learning process to be fixed in advance — usually backpropagation — and have taken the object of evolution to be the initial state of a system, upon which the learning process may act. The genesis of the learning mechanisms themselves is not investigated. This study will take a different approach. Here, the learning process itself will be the object of evolution. We will start with systems that are unable to learn, subject these systems to genetic recombination under selective pressures based on their ability to adapt to an environment within their lifetime, and see whether any interesting learning mechanisms evolve.

2 Evolution of Learning in Neural Networks

2.1 General Framework

The main idea in using genetic connectionism to model the evolution of learning is simple: use a genome to encode not the static properties of a network, but the dynamic properties. Specifically, the genome encodes a procedure for changing the connection strengths of a network over time, based on past performance and environmental information. In the experiment to be outlined here, the genome encodes no static properties at all. The initial configuration of a network is instead determined randomly (within a constrained framework). Thus, if genetic search is to be successful, it must evolve a learning procedure that can start from a wide variety of network configurations and reliably reach an appropriate result.

Working within this paradigm it is important to distinguish two levels of adaptation: learning — adaptation at the level of the individual — where an individual over time adapts to its environment, and evolution — adaptation at the level of the population — where over the course of evolutionary history a population gradually evolves mechanisms that improve the fitness of its members. In our case, the fitness of a single member of the population will be measured by its capacity for adaptation within its lifetime (learning); so the evolutionary mechanisms can be viewed as a second-order adaptive process: a population-wide adaptation that over generations produces better adaptive processes within the lifetime of an individual.

A vital component of the learning process is the *environment*. If the environment were relatively static, there might be little need for learning to evolve. Systems could instead evolve to a state where they have innate mechanisms to handle that environment. But if the environment is diverse and unpredictable, innate environment-specific mechanisms are of little use. Instead, individuals need general adaptive mechanisms to cope with arbitrary environments. In this way, a diverse environment encourages the evolution of learning.

Our basic *modus operandi* is as follows. A genome encodes the weight-space dynamics of a connectionist system — that is, it encodes a potential learning procedure. The fitness of a learning procedure is determined by applying it to a number of different learnable tasks. This is achieved by creating a number of networks and putting them in different environments for a specified amount of time. (In the experiments to be outlined here, an environment consists of a set of input patterns with associated training signals.) Each network has a random initial state, but its final state is determined by its interaction with the learning procedure and the environment. The fitness of the network is determined by how well it has adapted to its environment after the specified amount of time. The fitness of the learning procedure is derived from the fitness of the various networks that have used it. Learning procedures reproduce differentially over time, depending on their success in yielding adaptive networks. The hope is that from a starting population of essentially useless learning procedures, we will eventually produce something that enables powerful adaptation.

In principle, such a process could be of interest not only to biologists and psychologists but also to computer scientists and engineers. The space of possible learning mechanisms is very poorly understood. Genetic search allows the exploration of this space in a manner quite different from the usual combination of ingenuity and trial-and-error employed by algorithm designers. It is not impossible that genetic search could come up with a learning algorithm that rivals existing human-designed algorithms. At the very least, it might produce a learning algorithm already known to be useful. This, in fact, was the result of the experiments outlined here.

2.2 Implementation Details

We may now move from broad generalities to specific implementation details. In this preliminary experiment, an effort was made to keep things as simple as possible, in the interests of minimizing computational requirements and maximizing the interpretability of results. The choices made below were appropriate for a study of the feasibility of the methodology outlined above, but many of them could be varied with potentially quite different results.

2.2.1 Type of Learning Task

First, we must choose what kind of learning we will try to evolve. The three standard categories of learning tasks are supervised learning (learning with full training feedback about desired actions), reinforcement learning (where the only training information is a scalar representing the utility of a given action), and unsupervised learning (learning without any external teacher). Any of these kinds of learning could potentially be investigated via evolutionary methods, but here we chose supervised learning, as it is both the easiest and the best-understood of the three varieties. The tasks will involve associating specific input patterns with specific output patterns, where the desired output patterns are presented to the network as a training signal.

2.2.2 *Network Architecture*

The next choice to be made is that of the topology of the networks. Under the assumption that the genome codes for weight-space dynamics, we need not fix weight values in advance, but it is necessary that some form of network design be fixed. (Another approach might be to evolve network dynamics and network topology simultaneously, but that was not pursued here.) The simplest non-trivial topology is that of a single-layer feed-forward network. These networks also have the advantage that a powerful learning algorithm — the delta rule, also known as the Widrow-Hoff rule — is known to exist in advance, at least for supervised learning tasks. This experiment employs fully-connected single-layer feed-forward networks with sigmoid output units, with a built-in biasing input unit to allow for the learning of thresholds. A maximum connection strength of 20 was imposed, to prevent possible explosion under some learning procedures.

2.2.3 *Genetic Coding of Learning Mechanisms*

We need to be able to code complex forms of weight-space dynamics into a simple linear genome. Clearly, we are not going to be able to express all possible kinds of weight-space dynamics under a single encoding; instead, the dynamics must be severely constrained. In this experiment, it was decided that changes in the weight of a given connection should be a function of only information local to that connection, and that the same function should be employed for every connection. For a given connection, from input unit *i* to output unit *j*, local information includes four items:

a_j	the activation of the input unit *j*.
o_i	the activation of the output unit *i*.
t_i	the training signal on output unit *i*.
w_{ij}	the current value of the connection strength from input *j* to output *i*.

The genome must encode a function *F*, where

$$\Delta w_{ij} = F(a_j, o_i, t_i, w_{ij}).$$

It was decided that F should be a linear function of the four dependent variables and their six pairwise products. Thus F is determined by specifying ten coefficients. (Note that this framework for weight-space dynamics does not exclude the delta rule as a possible candidate. This is not, of course, entirely coincidental, and the charge of using prior knowledge about learning to constrain the evolutionary process might be leveled. However, one advantage of the network topology we have chosen is that a sufficiently simple good learning procedure exists. Due to the simplicity of the procedure, we can allow it as a possibility without rigging the possibilities in too arbitrary a manner in advance.)

The genome specifies these ten coefficients directly, with the help of an eleventh "scale" parameter. We put

$$\Delta w_{ij} = k_0(k_1 w_{ij} + k_2 a_j + k_3 o_i + k_4 t_i + k_5 w_{ij} a_j + k_6 w_{ij} o_i + k_7 w_{ij} t_i + k_8 a_j o_i + k_9 a_j t_i + k_{10} o_i t_i).$$

The genome consists of 35 bits in all. The first five bits code for the scale parameter k_0, which can take the values 0, ±1/256, ±1/128, ..., ±32, ±64, via exponential encoding. The first bit encodes the sign of k_0 (0=negative, 1=positive), and the next four bits encode the magnitude. If these four bits are interpreted as an integer j between 0 and 15, we have

$$|k_0| = \begin{cases} 0 & \text{if } j = 0 \\ 2^{j-9} & \text{if } j = 1, \ldots, 15. \end{cases}$$

The other 30 bits encode the other ten coefficients in groups of three. The first bit of each group expresses the sign, and the other two bits express a magnitude of 0, 1, 2 or 4 via a similar exponential encoding. If we interpret these two bits as an integer j between 0 and 3, then

$$|k_i| = \begin{cases} 0 & \text{if } j = 0 \\ 2^{j-1} & \text{if } j = 1, 2, 3. \end{cases}$$

For example, the delta rule could be expressed genetically as:

11011 000 000 000 000 000 000 000 010 110 000

where the coefficients decode to

4 0 0 0 0 0 0 0 –2 2 0

and the appropriate formula is thus

$$\begin{aligned}\Delta w_{ij} &= 4(-2a_j o_i + 2a_j t_i)\\ &= 8a_j(t_i - o_i).\end{aligned}$$

2.2.4 *The Environment*

Given that we are trying to evolve successful performance on supervised learning tasks, we need an appropriate environment consisting of a number of learnable tasks. Each "task" in the environment used here consists of a mapping from input patterns to output patterns. As we are using single-layer networks, learnable mappings must be linearly separable. Thirty diverse linearly separable mappings were used. These had between two and seven units in the input patterns, and always a single unit as output. (Any single-layer network with more than one output unit is equivalent to a number of disjoint networks with a single output unit each, so there is no point using more than one output unit at a time.) For each task, a network was presented with a number of training exemplars, each consisting of an input pattern and associated output.

a_1	a_2	a_3	a_4	a_5	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
1	1	1	1	1	1	0	0	0	1	0	1	0
0	0	0	0	0	0	0	1	0	0	1	0	1
0	1	1	1	0	0	0	1	0	1	0	1	0
1	1	0	0	0	0	1	1	1	0	1	1	1
1	0	1	0	1	1	0	0	0	0	0	1	1
0	1	1	0	0	0	0	1	0	1	1	0	1
0	1	1	1	1	1	0	1	0	1	0	1	0
0	1	0	0	0	0	0	1	1	1	1	0	1
1	1	0	0	1	1	0	0	1	0	1	1	1
1	0	0	1	0	0	0	1	1	1	0	1	1
1	0	1	1	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	0	0	1

Table 1. Eight tasks from the environment.

Table 1 shows eight of the tasks that were used. Each of these was a linearly separable mapping from five input units to a single output unit. The eight rightmost columns of the table represent the correct outputs for the eight different tasks. The five leftmost columns represent the input units, common to all eight tasks. For each task, twelve training exemplars were presented, corresponding to the twelve rows of the table. For example, the first task shown was to detect whether the fifth unit in the input pattern was on or off. The second task involved recognizing a single specific pattern. The other tasks were more complex.

2.2.5 Evaluation of Fitness

With this diverse "environment" in hand, evaluation of the fitness of a given learning algorithm proceeds as follows. A number of tasks (typically 20) are randomly selected at the start of every evolutionary run. For each learning algorithm in the population, we measure its performance on each task. Fitness on a given task is measured by the following procedure:

(1) Create a network with the appropriate number of input units for the task and a single output unit.

(2) Initialize the connection strengths of the network randomly between –1 and 1.

(3) For a number of epochs (typically 10), cycle through the training exemplars for the task, where for each exemplar we:

> (3a) Propagate input values through the system, yielding output values; then
>
> (3b) Adjust the weights of the system according to the formula specified by the learning procedure, on the basis of inputs, output, training signal, and current weights.

(4) At the end of this process, fitness on the task is measured by testing the network on all training exemplars, and dividing the total error by the number of exemplars, subtracting from 1, and multiplying by 100. This yields a fitness "percentage" between 0 and 100.

Fitness of the learning rule is obtained by evaluating its performance on each of the (typically 20) tasks, and taking the mean fitness over all tasks. In this way every learning procedure is assigned a fitness between 0 and 100%. A fitness of 50% represents chance performance — i.e., no learning over the lifespan of an individual network. A fitness of 100% indicates perfect learning, at least on the given tasks: at the end of ten epochs, each mapping is learned perfectly. It should be noted that given the limited number of epochs, perfect fitness seems unachievable if we are using more than one or two tasks: even the delta rule has a fitness of only around 98%.

There is some stochastic variation in the fitness of a given learning rule, due to the random variation in initial weights of the networks. This variation is reduced by maintaining a history of each specific learning rule that appears in a given evolutionary process. If a learning rule has appeared already in an evolutionary run, its fitness is computed by the above procedure, but the fitness used for purposes of selection is the mean fitness from this and all prior appearances. This occurs until a given learning rule has appeared fifteen times, at which time the mean fitness is recorded for good and never recomputed, thus saving a significant amount of computational resources.

2.2.6 Parameters of the Genetic Algorithm

For those unfamiliar with the genetic algorithm, it works as follows. We start with a population of appropriate genomes — in this case random bit-strings of 35 bits each. In these experiments, the population size was always 40. Every generation, each genome is converted into a phenotype (here, a learning procedure), and its fitness is measured (as outlined above). Then we have a process of differential *reproduction*. Each genome probabilistically makes copies of itself, with the probability of reproduction being linearly proportional to its fitness. This yields a new population of 40 genomes, which is then subjected to the process of *crossover*: pairs of genomes are "mated" by taking a randomly selected string of bits from one and inserting it into the corresponding place in the other, and vice versa, thus producing two new genomes. In this experiment, 80% of the population are subject to this process, while the remaining 20% reproduce asexually by copying themselves directly into the new population. "Elitist" selection is also used: the best individual in the previous population is guaranteed a place in the new population by asexual reproduction. Finally, in the resulting population of 40 genomes of 35 bits each, each bit has a 1% chance of *mutation* — that is, of changing from 0 to 1 or vice versa. This process of fitness-measurement, reproduction, crossover and mutation is repeated for a number of generations, usually 1000.

Summary of genetic algorithm parameters: Population = 40, two-point crossover and elitist selection are used, crossover rate = 0.8, mutation rate = 0.01, number of generations = 1000.

3 Results

3.1 Results of initial evolutionary runs

When the genetic algorithm is run, initial results are as expected. At the start of the run, all individuals present in the population have fitness between 40% and 60%, indicating no significant learning behavior. Instead, weight-dynamics are essentially random over time. Within a few generations, differential reproduction begins to exploit even small differences in fitness, and the fitness of the best individuals in the population rises rapidly as simple adaptive mechanisms make their way into the weight-space dynamics. After 1000 generations, the maximum fitness is typically between 80% and 98%, with an mean of about 92%. Table 2 gives the results of ten separate runs with the same parameters but different random seeds.

k_0	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_9	k_{10}	Fitness
0.25	0	0	0	0	0	0	0	-4	4	0	89.6%
-2.00	0	0	0	0	0	0	0	2	-2	0	98.0%
0.25	0	-1	-2	4	0	0	0	-2	4	-2	94.3%
0.25	0	-1	-2	4	0	0	0	-2	4	-2	92.9%
-0.25	0	0	1	-1	0	1	-1	4	-4	0	89.8%
-1.00	0	0	-1	1	0	0	0	4	-4	0	97.6%
4.00	0	0	1	-1	0	0	0	-2	2	0	98.3%
-0.06	0	0	0	-2	-1	2	2	4	-4	2	79.2%
-0.25	0	0	2	-1	0	-1	-1	2	-4	0	89.8%
0.25	0	-1	-2	4	0	0	0	-2	4	-2	93.2%

Table 2. Best learning algorithms produced on 10 evolutionary runs.

Note that even the worst learning rule has significant adaptive ability, with a fitness of 79.2%. This rule, then, enables a network to predict the correct output for a given input with approximately 80% accuracy after 10 epochs of training. Other rules are significantly better, with a mean fitness of 92.3%.

On the second of the ten runs, the genetic search process discovered a version of the well-known delta rule (or Widrow-Hoff rule). The learning rule here was:

$$\begin{aligned} \Delta w_{ij} &= -2(2a_j o_i - 2a_j t_i) \\ &= 4a_j(t_i - o_i). \end{aligned}$$

This rule unsurprisingly has a high fitness of 98.0%. Such an event was not unusual — the delta rule was discovered on perhaps 20% of all runs with similar parameters. In this set of ten runs, the delta rule evolved twice (the second occurrence has a much lower value of the "learning rate" k_0, and so a lower fitness of 89.6%). Slight variations on the rule also evolved twice, with high fitnesses of 98.3% and 97.6%. In the seventh run above, the rule is

$$\begin{aligned}\Delta w_{ij} &= 4(o_i - t_i - 2a_j o_i + 2a_j t_i) \\ &= 8(a_j - 0.5)(t_i - o_i).\end{aligned}$$

In the sixth run above, the rule is

$$\begin{aligned}\Delta w_{ij} &= -1(-o_i + t_i + 4a_j o_i - 4a_j t_i) \\ &= 4(a_j - 0.25)(t_i - o_i).\end{aligned}$$

The similarity of these rules to the delta rule is clear.

Typical evolutionary progress in fitness is shown in Figure 1. This is a graph of the fitness of the best individual learning rule in the population every 25 generations over the course of the run. The figures are averaged over the ten runs mentioned above.

3.2 Effect of Environmental Diversity on the Evolution of Learning

Given the learning algorithms that this evolutionary method produces, it is natural to ask whether the algorithms are adapted specifically for the tasks that have been present in the environment during the evolutionary process, or whether the algorithms are in fact very general adaptive mechanisms capable of learning a wide range of tasks, including tasks that were not present during the evolutionary process. The delta rule clearly falls into the second category of general adaptive mechanisms, but for other kinds of weight-dynamics evolved by this process, the answer is not so clear *a priori*.

It might well be thought that given the limited number of tasks in the evolutionary environment, the evolved learning mechanisms would function well only on those tasks. At the very least, it is plausible that performance on such tasks might be superior to performance on

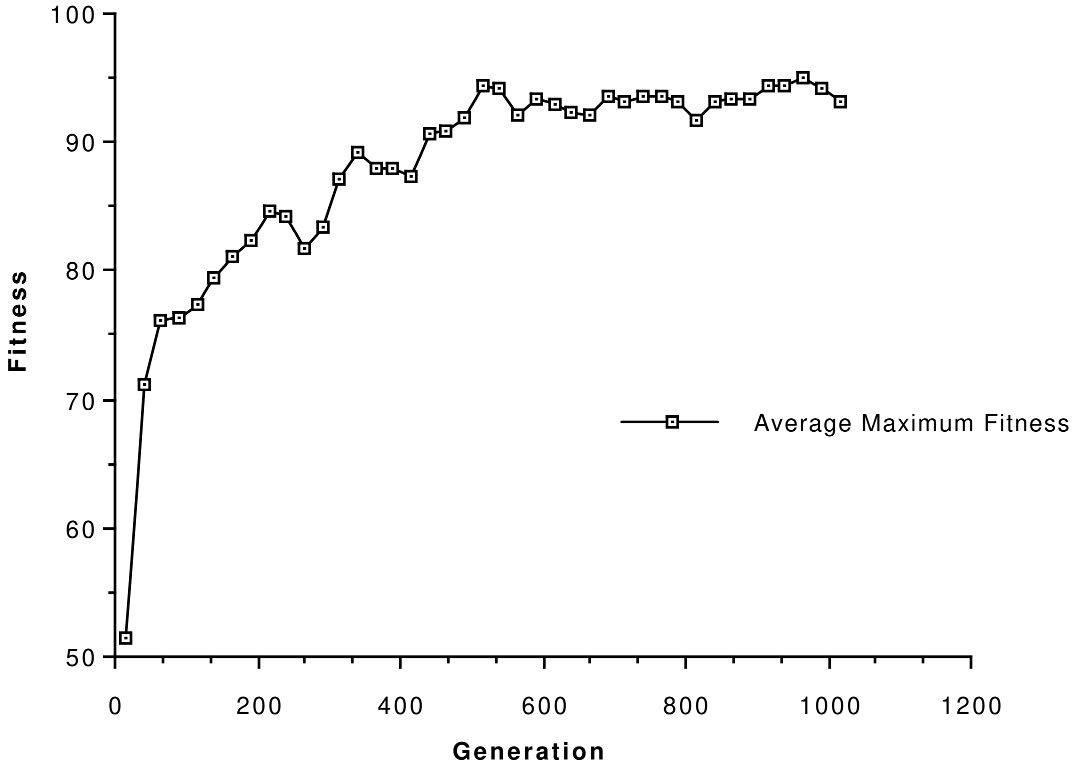

Figure 1. The evolution of maximum fitness in the population.

tasks not in the evolutionary environment. On the other hand, it seems plausible that if enough tasks are present in the evolutionary environment, then it is unlikely that highly task-specific mechanisms would evolve. In the design of the experiment, 20 different tasks were used in the environment of a given evolutionary run precisely because this seemed a large enough number of tasks to minimize this likelihood.

The task-specificity of evolved learning mechanisms can be measured by applying such mechanisms to a selection of new tasks not present in the evolutionary environment. In this experiment, this measurement was made by starting from a pool of 30 tasks. On every run, 20 of these were randomly selected and designated as part of the evolutionary environment, and evolution proceeded on the basis of how well these 20 tasks could be learned. The other 10 tasks were designated as test tasks: at the end of an evolutionary run, the best learning algorithm in the evolved population was tested on these 10 new tasks, and mean fitness was measured.

For the 10 runs outlined in the previous section, the average fitness of the 10 final learning algorithms on the 20 tasks in their evolutionary environment was 92.3%. The average

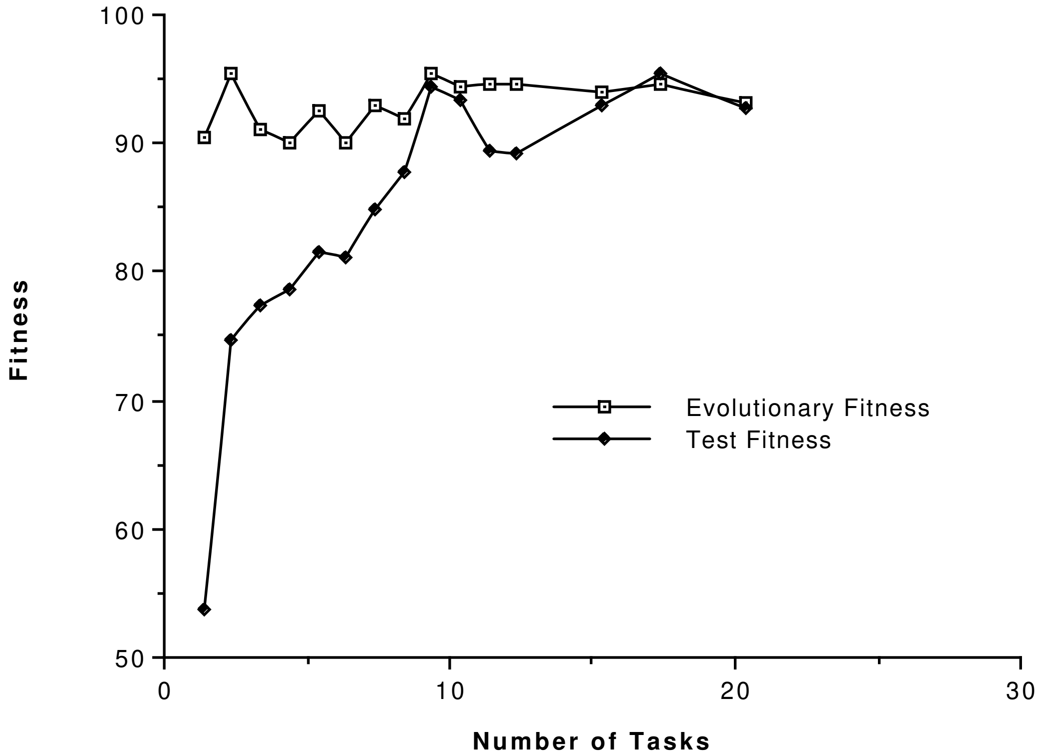

Figure 2. Evolutionary fitness and test fitness, versus number of tasks.

of these algorithms on the tasks not present in the evolutionary environment was 91.9%. This indicates that the evolutionary environment was sufficiently diverse that there was no significant tendency for task-specific mechanisms to evolve.

The next question of interest is that of how diverse the environment must be to force the evolution of general learning mechanisms. To investigate this, the number of tasks in the evolutionary environment was varied between 1 and 20 on a number of runs. Performance of a learning algorithm on this set of tasks is referred to as *evolutionary fitness*. After 1000 generations, the learning algorithm with the highest evolutionary fitness was removed and tested on 10 further tasks that were not in the evolutionary environment. Performance on this set of tasks — the *test fitness* — indicates the generality of the learning mechanisms that evolve. For each of a number of values of the number of tasks in the evolutionary environment, 10 runs were performed with different random seeds, and the results for these runs were averaged. Figure 2 graphs (1) the average final evolutionary fitness and (2) the average test fitness, as a function of the number of tasks in the evolutionary environment.

The first thing to note is that evolutionary fitness is fairly constant as a function of the number of tasks, always somewhere around 90%. In some ways, this is surprising — we might expect superior performance on a smaller number of tasks. It seems plausible that it would be easier to evolve good learning mechanisms for one task than for 20. The reason for the observed constancy perhaps lies with the fact that when there is only a single task, there are more ways to learn to perform it quite well but suboptimally. The greater ease of finding a successful learning mechanism may be counterbalanced by an increased chance that the evolutionary process will get stuck in a suboptimal local maximum, from which it is difficult to escape even by genetic recombination.

The more important thing to note is that test fitness does indeed decrease significantly as the number of tasks decreases. When there is a single task in the evolutionary environment, test fitness averages 53%, or only just above chance. This indicates that the kind of weight-dynamics that evolves, while performing well on the task in the evolutionary environment (fitness 89.7%), is not suited at all for other tasks. A task-specific mechanism has evolved, rather than a general learning mechanism. When the number of tasks is between 1 and 9, test fitness rises above chance but is well below evolutionary fitness, again indicating some degree of task-specificity in the learning mechanisms. When the number of tasks is about 10, test fitness rises to a value very close to evolutionary fitness, indicating that this number of tasks is sufficient to force the evolution of general learning mechanisms.

The moral here is clear: a sufficiently diverse environment brings about the evolution of general learning mechanisms. This makes intuitive sense. If the environment is fairly constant, there is no need for evolution to produce mechanisms that function in more general contexts — instead, we might expect organisms to inherit innate mechanisms adapted to that specific environment. When the environment is diverse and relatively unpredictable, we should expect individual organisms to inherit adaptive mechanisms capable of coping with a variety of different environmental features as they arise. This is precisely what we have found here.

These results also have some bearing on the traditional controversy between nativists and empiricists. When the environment in these experiments was not diverse, the mechanisms that evolved to cope with that environment may be regarded as *innate*, at least in a weak sense. These mechanisms are present specifically to cope with certain tasks. Although the capacity to deal directly with the tasks is not actually present at the beginning of the lifetime of an individual, the capacity is triggered by an expected sequence of events from the environment on the basis of task-specific information present in the individual's genome. In this case, we might say that the weight-space dynamics represent a task-specific *developmental* process for an innate capacity, rather than a learning process. When the environment is diverse, on the other hand, the only mechanisms that individuals possess innately are general learning mechanisms. In a situation familiar to empiricists, the individual is a tabula rasa, ready to be imprinted with whatever regularities it finds in the environment.

We should be cautious about applying the results from these limited experiments to theorizing about natural biology and psychology. In particular, an important difference is that in these experiments, genetic information codes only dynamic properties of an individual, and specifies no structural information about the individual's initial state. In natural biological systems, the genome appears to carry much information about the initial structure of a system. This would make some differences to our conclusions about innateness, for example. But a general moral may still be drawn, if strewn with caveats: innate mechanisms are likely to be the product of a relatively constant evolutionary environment; adaptive, empiricist-style mechanisms may be a consequence of environmental diversity.

4 Discussion and Future Directions

We have seen that the methods of genetic search and connectionism can be combined to provide a demonstration that the capacity for learning (adaptation within the lifetime of an individual) may evolve in a natural fashion under the pressures of evolution (adaptation over the history of a population). This double adaptive loop seems to be a powerful mechanism for coping with a diverse environment. All that seems necessary for learning to evolve in this fashion is that the genome encode the long-term *dynamics* of the information-processing capacities of an individual. If genetic variation allows for variation in these dynamics, then a diverse environment will exert pressure towards dynamics that are adaptive over the lifetime of an individual; eventually, we should expect learning mechanisms to arise.

The kind of learning that we have allowed here has been very simple, to allow a simple proof of possibility and an illustration of the key issues. Supervised learning is a fundamental kind of learning, but it is not very plausible biologically, and it also suffers from the problem that we may understand it *too* well. At least in feed-forward networks, we already possess very powerful supervised learning methods in the delta rule and its generalization to multi-layer networks, backpropagation. It is unlikely that evolutionary methods will allow us to improve much on these methods. So while the methods here can yield some insight into the evolutionary processes behind learning, they have not given much insight into learning itself.

Other forms of learning are much less well-understood, however, and the methods of genetic connectionism outlined here may provide a novel means of insight. For example, reinforcement learning is much more difficult than the supervised variety. A number of learning algorithms are known (e.g. Barto 1985, Williams 1988), but none have been as successful as backpropagation has been. Using genetic search, we can investigate the space of possible dynamics of reinforcement learning systems, and it is not impossible that we might come up with novel algorithms. The extension, in principle, is a simple one. The problem of unsupervised learning, although more complex, could be attacked in a similar fashion.

Another possible generalization would be to a different class of network architectures. We could, for example, attempt to evolve learning algorithms for recurrent network. Another interesting approach would be to attempt to evolve static and dynamic properties of a network simultaneously. Network topology, the values of certain weights, and a learning algorithm could be simultaneously encoded by a genome, and genetic search might fruitfully explore the interaction between these factors.

There is one problem that is always lurking in the background here, and that is the problem of the genetic coding. How do we choose such a coding? Do we try to allow for as many kinds of weight-space dynamics as possible, or do we constrain the space using prior knowledge? How could we possibly find a coding of possible dynamics that includes as possibilities all the diverse learning algorithms proposed by humans to date? In the experiment described in this paper, the decision was easy. The networks were small, there was only a certain amount of relevant information, and it was known in advance that a simple quadratic formula could provide a good learning mechanism. The encoding of more ambitious mechanisms may not be so simple. To attempt to evolve backpropagation by this method, for example, we would need either a highly complex genetic coding, or else a simple but very specific coding that was rigged in advance to allow backpropagation as a possibility. When we do not know the form of a plausible learning algorithm in advance — and this is the most interesting and potentially fruitful application of these methods — the problem of the coding becomes very important. Only so much can be coded into a finite-length bit string.

One way around the limitation of prespecified codings of dynamic possibilities would be to move away from the encoding of learning algorithms as bit-strings, and instead encode algorithms directly as function trees. In a recent report, Koza (1990) has demonstrated the potential of performing genetic-style recombination upon function-tree specification of algorithms. This method of “genetic programming” uses recombination and selection in a fashion very similar to traditional genetic methods, but with the advantage that under evolutionary pressures such function-trees may become arbitrarily complex if necessary. This open-endedness may be a good way of getting around the limitations inherent in fixed genetic codings. Furthermore, the method is a very natural way of encoding dynamic, algorithmic processes of the kind we are investigating here.

Even if we are forced to constrain the possible kinds of learning algorithm, genetic methods provide a powerful way of searching the spaces of such learning algorithms. Unlike other phenotypic spaces such as those of classifier systems or network topologies, the space of learning algorithms is so poorly understood that even the gross qualitative properties of a given algorithm are very difficult to predict in advance. Genetic search may allow to us uncover algorithms that humans might never consider.

In sum, genetic connectionism provides a tool of analysis that may be of interest to biologists and psychologists, and also to computer scientists and engineers. Genetically-based methods provide a direct way to model evolution, a powerful method of search, and a paradigm of emergence-over-time. Connectionist methods have the potential for sophisticated forms of learning via their paradigm of emergence-over-levels. Combining genetic emergence-over-time with connectionist emergence-over-levels seems to provide a property that neither class of methods possesses alone: automated creativity in the design of adaptive systems.

Acknowledgements

Thanks to Liane Gabora, Gary McGraw and Bob French for helpful comments.

References

A. G. Barto (1985). Learning by statistical co-operation of self-interested neuron-like computing elements. *Human Neurobiology*, 4: 229–256.

R. Belew, J. McInerney & N. N. Schraudolph (1990). Evolving networks: Using the genetic algorithm with connectionist learning. CSE Technical Report CS90–174, University of California, San Diego.

D. E. Goldberg (1989). *Genetic algorithms in search, optimization and machine learning*. Reading, MA: Addison-Wesley.

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

J. H. Holland (1975). *Adaptation in natural and artificial systems.* Ann Arbor: University of Michigan Press.

J. H. Holland (1986). Escaping brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems. In J. Michalski, J. G. Carbonell & T. M. Mitchell (eds.) *Machine Learning II*. Los Altos, CA: Morgan Kaufmann.

J. H. Holland, K. J. Holyoak, R. E. Nisbett, & P. R. Thagard (1986). *Induction: Processes of Inference, Learning and Discovery*. Cambridge, MA: MIT Press.

J. Koza (1990). Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Technical Report STAN–CS–90–1314, Stanford University.

G. Miller, P. Todd & S. Hegde (1989). Designing neural networks using genetic algorithms. In *Proceedings of the Third Conference on Genetic Algorithms and their Applications*, San Mateo, CA: Morgan Kaufmann.

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

J. Maynard Smith (1987). When learning guides evolution. *Nature*, 329: 761–762.

D. Whitley, T. Starkweather & C. Bogart (1990). Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel Computing*, forthcoming.

S. Wilson (1990). Perceptron redux. *Physica D,* forthcoming.

R. J. Williams (1988). Toward a theory of reinforcement-learning connectionist systems. Technical report NU–CCS–88–3, Northeastern University.