*Evolutionary Computation* 4(3):235-270, 1996.

# Modeling the Evolution of Motivation

John Batali* and William Noble Grundy†

**Abstract**

In order for learning to improve the adaptiveness of an animal's behavior and thus direct evolution in the way Baldwin suggested, the learning mechanism must incorporate an innate evaluation of how the animal's actions influence its reproductive fitness. For example, many circumstances that damage an animal, or otherwise reduce its fitness are painful and tend to be avoided. We refer to the mechanism by which an animal evaluates the fitness consequences of its actions as a "motivation system," and argue that such a system must evolve along with the behaviors it evaluates.

We describe simulations of the evolution of populations of agents instantiating a number of different architectures for generating action and learning, in worlds of differing complexity. We find that in some cases, members of the populations evolve motivation systems that are accurate enough to direct learning so as to increase the fitness of the actions the agents perform. Furthermore, the motivation systems tend to incorporate systematic distortions in their representations of the worlds they inhabit; these distortions can increase the adaptiveness of the behavior generated.

*Department of Cognitive Science; University of California at San Diego; 9600 Gilman Drive; La Jolla, CA 92903-0515; (619) 534-7308; `batali@cogsci.ucsd.edu`

†Department of Computer Science and Engineering; University of California at San Diego; 9600 Gilman Drive; La Jolla, CA 92903-0114; (619) 453-4364; `bgrundy@cs.ucsd.edu`

# 1 Introduction

In his presentation of what he calls "a new factor" in evolution, James Mark Baldwin (1896) points out that "those congenital or phylogenetic adaptations are kept in existence, which tend themselves to intelligent, imitative, adaptive, and mechanical modification during the lifetime of creatures which have them." The ability to undergo such "ontogenetic adaptation," Baldwin argues, allows evolution to follow "certain lines of determinate phylogenetic variation in the directions of the determinate ontogenetic adaptation of the earlier generation."

In this way, evolution and the various "ontogenetic agencies" Baldwin discusses — in particular, learning — can work synergistically. Those animals capable of improving their behavior by learning will have an advantage over those that cannot, and those better able to improve will be at a correspondingly greater advantage. Furthermore, the advantages of possessing the specific improvements that learning enables can also result in the selection of those animals that possess the improved traits innately. Therefore, the species will evolve "in the directions thus ratified by intelligence," even though there is no inheritance of acquired characteristics.

"How," Baldwin then asks, "does the organism secure, from the multitude of possible ontogenetic changes which it might and does undergo, those which are adaptive?" His answer is that it is a

> ... fact of physiology ... that the organism concentrates its energies upon the locality stimulated, for the continuation of the conditions, movements, stimulations, which are vitally beneficial, and for the cessation of the conditions, movements, stimulations, which are vitally depressing and harmful.

The pleasure received from such "beneficial" stimulations results in their being repeated, and ultimately made habitual:

> This form of concentration of energy upon stimulated localities, with the resulting renewal by movements of conditions that are pleasure-giving and beneficial, and the subsequent repetitions of the movements is ... the adaptive process in all life, and this process is that with which the development of mental life has been associated.

This proposal closely resembles the "law of effect," suggested contemporaneously by Thorndike (1898), which states that animals will tend to repeat those actions that lead to "satisfying" states of affairs and avoid those that lead to "annoying" ones.

Of course Baldwin was not the first to suggest that feelings of pleasure and pain often motivate animal behavior. Aristotle makes the capacities of sensation and movement the defining properties of animals and posits that expectations of pleasure and pain guide their movements. (See Hamlyn, 1968; Robinson, 1989.)

Often left unspoken in such proposals is the observation that the capacity to experience feelings of pleasure and pain is quite an evolutionary achievement. Baldwin, in a passage cited above, conjoins "pleasure-giving" and "beneficial," but of course the two are not equivalent. Pleasure and pain are not objective features of the physical environment. Somehow the animal must evolve the tendency to feel pleasure in those conditions that are beneficial to its reproductive fitness and pain in those that are deleterious. Some mechanism that can evaluate how the animal's actions influence its fitness must underlie these feelings of pleasure and pain.

We refer to this mechanism as a "motivation system" because the animal will, as a result of learning, tend to perform those actions which are favorably evaluated by the system. The motivation system may also be involved in the selection of actions, with the animal performing those actions that it expects to be favorably evaluated. In either case, the evaluations of the motivation system must be at least somewhat correlated with the actual fitness consequences of the animal's actions for its behavior to be adaptive.

Such an evaluation is in general not a simple calculation. Many physiological and environmental factors can be involved, and can interact in intricate ways. Subtle changes in the animal's physiological state or its environment can have extreme fitness consequences, especially for animals and environments of any significant complexity.

Researchers writing about motivation disagree about the definition of the term. Most admit that motivation involves causation — the initiation of and persistence in performing actions. However, Toates (1986) and, to a lesser extent, Colgan (1989) include goal-directedness as an aspect of motivation. Toates contrasts this cognitive definition of motivation with the behaviorist one offered by Hull (1952), in which learning theory accounts for purposive action, leaving motivation to describe more immediate drives. Like Toates, Epstein (1982) treats motivation cognitively. According to Epstein, the affective expectancy of motivation is a primary characteristic distinguishing it from instinct.

In what follows, we remain uncommitted concerning the cognitive nature of motivation, primarily because the simplicity of the models to be discussed precludes any but metaphorical applications of cognitive terms. Some of the models may be considered to be goal-directed, but only in a very limited sense, since they make predictions only about the immediate future.

Our definition of motivation coincides most strongly with that of McFarland and Bösser (1993) who describe motivation systems in terms of "expected utility." McFarland and Bösser go on to discuss planning as an essential aspect of motivation. Again, we omit such discussion only because of the simplicity of our models.

Whether we speak of fitness consequences or expected utility, the required evaluation requires an innate component. If an animal's behavior is modified by learning, the learning must itself be guided by an evaluation of the fitness consequences of the modifications performed, that is, by another motivation system. In order to avoid an infinite regress, at least one such system cannot be learned.

Animals whose actions are guided by a motivation system will be under strong selection pressure for those systems to be accurate. If its motivation system incorrectly favors actions with negative fitness consequences, an animal will tend to perform those actions and will be less likely to pass its genes, including those specifying its motivation system, to subsequent generations.

In real animals, the motivation system is highly distributed, incorporating aspects of the animal's nervous and sensory systems as well as its physiology and biochemistry (Colgan, 1989, chapter 3; Thompson, 1986). This is not surprising, since motivation systems profoundly affect the evolution of adaptive behavior in even the simplest organisms.

Motivated by these issues, we explore the questions of whether and how motivation systems can evolve, how they can be incorporated into the generation of adaptive behavior, and how the innate and learned components of mechanisms for generating behavior relate to each other.

Our explorations involve computational simulations of the evolution of simple agents whose reproductive fitness is based on the actions they perform in simple worlds. An agent can affect the world by its actions, and some effects, in some states of the world, are better for the agent's fitness than others. Some of our agents contain learning components that modify the agents' behaviors as they interact with the world.

We look for effects of the sort Baldwin described, in particular, for simulations in which agents learn adaptive behaviors in early generations, but for whom those behaviors become increasingly innate in later generations. We find, in some of our worlds, that populations of learning agents outperform populations of agents whose behavior is innate, and learning enables such agents to acquire increasingly adaptive innate behaviors as the simulations progress. On the other hand, not all of the simulations showed such results. Details of the worlds, and of the agents' architectures, strongly influence the evolutionary dynamics of populations of such agents.

When our populations are successful in evolving motivation systems, such systems are able to evaluate the fitness consequences of actions with fairly high accuracy. However, the motivation systems tend to incorporate systematic distortions in their representations of the worlds they inhabit. These distortions often increase the likelihood that the behavior of the agents is adaptive. They do so by exploiting regularities in the world, facilitating the learning of important features of the worlds, and encoding adaptive behavioral strategies.

In the next section we summarize related research. Section 3 presents our abstract model of worlds and the difficulty measures that apply to them. Section 4 describes the agent architectures we use and the algorithms they incorporate for action selection and learning. Our evolutionary simulation methodology is explained in section 5. Section 6 presents the results of our simulations; these results are compared and discussed in section 7.

## 2 Related Work

A number of other research projects have explored the relations between learning and evolution, and our investigations are inspired by, and build on, their results.

Hinton and Nowlan (1987) present a succinct model of the Baldwin effect, which demonstrates how, in their model at least, changes in an organism during learning can affect the evolutionary dynamics of populations of organisms capable of learning. However, like Baldwin, Hinton and Nowlan do not account for the evolution of the motivation system that evaluates the fitness consequences of changes to the organism. Indeed, in the Hinton and Nowlan model the evaluation was performed by the experimenter, who not only determined the fitness of genotypes, but also determined when the learning mechanism had reached the solution and should cease its operations. In this regard, the model of Hinton and Nowlan resembles a number of experiments in which populations of neural networks are incorporated into some variety of evolutionary computations and selected according to their performance after training on some fixed task (Belew, 1989; Chalmers, 1991; Belew, et al., 1991; Batali, 1994).

Littman (1996) describes several projects in which the evolution of something like a motivation system is more explicitly addressed. For example, Ackley and Littman (1992) describe the experimental paradigm ("Evolutionary Reinforcement Learning") that we adopt — a population of agents with two evolved neural networks, one that generates actions, and the other that evaluates results of the actions and controls the training of the first network. However, in Ackley and Littman's experiment, the motivation evaluation is fairly simple, since the evaluation network is given input signals representing the "health" and "energy" of the agent. These two quantities determine which agents reproduce and which die. We are interested in whether the agents can evolve to correctly evaluate the fitness of consequences of actions given much less easily interpretable input.

Todd and Miller (1991) describe simulations of entities capable of responding to a number of sensory signals, and which evolve the ability to detect features of the environment correlated with fitness. The agents in these simulations consist of very simple neural networks, each containing only three nodes. Yet even these simple agents, given a task appropriately tailored for their architecture, manage to both learn and evolve. In addition to illustrating an interesting U-shaped relationship between the use of the internally generated reinforcement and the accuracy of that reinforcement, Todd and Miller's work provides an example of a simulation in which the learning architecture and the learned task are closely linked.

Nolfi and Parisi (1993) describe simulations in which learning agents evolve internal "auto-teaching" nodes. These nodes, which are integrated with the agents' neural network architectures, evolve to approximate a teaching input of the sort used in supervised learning algorithms. Although Nolfi and Parisi's agent architecture differs significantly from the ones described here, some of their results, including the gradual transformation from learned behaviors to innate behaviors and the unpredictable but nonetheless useful signals generated by the auto-teaching nodes, coincide with ours.

Each of these projects was carried out in a single world with one or two architectures, and each experiment employed a relatively simple fitness function. One primary goal of our work is to begin to systematically explore various alternative learning architectures and tasks. Accordingly, in addition to describing learning architectures, we need an abstract framework with which to describe learning tasks. That framework is the subject of the next section.

# 3 Worlds

Our analysis and experiments use an abstract model of simple worlds in which a solitary agent's actions affect its environment and its reproductive fitness.

A number of existing models of the evolution of behavior and the interactions between learning and evolution make use of fanciful world models in which a two-dimensional grid representing physical space is populated by "food" and "poison" and "predators" and so forth. The available actions involve moving around in the space, eating the food or the poison, and avoiding or killing the predators. While such models have a certain intuitive appeal, and have led to deep insights about the evolution of behavior, it is often unclear the degree to which details of the models influence the results of evolutionary simulations involving those models. We developed the abstract world model presented here so that we could systematically explore such issues by varying world models and agent architectures. We can, for example, characterize different worlds in terms of measures of difficulty, and see how the characterization of the difficulty of a world influences how well different agent architectures can evolve to behave in it, as well as how learning and evolution interact in that world.

## 3.1 A World Model

In our model, a **world** is characterized as a set $E$ of **environments**, a set $A$ of **actions** available to an agent in that world, and two functions:

$$\begin{aligned} \mathit{physics}\,(E \times A) &\to E \\ \mathit{fitness}\,(E) &\to R \end{aligned}$$

The **physics function** takes an environment–action pair to the environment that results from performing the given action in the given environment. The **fitness function** takes an environment to a real number representing the fitness increment due to being in that environment. In general, both of these functions could be stochastic, though we focus on deterministic (and very simple) worlds in this paper.

Intuitively, an animal's actions can affect the external world as well as its internal state (for example, by eating something). In our model, we do not explicitly differentiate between the internal and external environments of an agent. There should be no way, *a priori*, for the agent to determine which aspects of the environment affect fitness. Providing an explicit delineation of the agent's body from the rest of the environment would constitute a strong clue as to which environmental features are relevant to fitness.

Despite its highly abstract nature, this framework can model many of the types of worlds used in other simulations of the evolution of behavior. For example, environment states may be interpreted as positions of the agent on a grid, together with the presence of food, predators, and other items, and perhaps features like the health of the agent. Actions can be interpreted as motion on the grid, or as ingesting items, each with fitness consequences that depend on the environment in which it is performed.

## 3.2 Three Worlds

Each of the three simple worlds we used in our experiments has sixteen different environment states and four actions. The physics and fitness functions of the worlds are specified in figure 1, and are presented in tabular form in table 1.

The physics function of the world **count-flip** is obtained by inverting a bit in the binary representation of the environment's number; the bit that is flipped depends on the action chosen. The fitness function of the count-flip world depends on the number of 1's in the binary representation of the environment's number. If the number of 1's is 3 or 4, the fitness of the state is $-1$. If the number of 1's is 2, the fitness of the state is $+1$. Otherwise, the fitness of the state is 0. This world is designed to be relatively easy for a neural network to master — a good action strategy in this world would be to count the number of 1's in the current environment; If that number is low, an action that flips a 0 to 1 should be performed; If the number of 1's is high, a 1 should be flipped to a 0. (More precise characterizations of the "difficulty" of the worlds will be described shortly.)

The physics function of the world **mod-inc** is obtained by adding the number of the action to the number of the current environment and taking the result modulus the total number of environment states. The fitness function of mod-inc is obtained by computing the number of the state modulus 3 and subtracting 1 from the result. This world is designed to be relatively difficult, as both the physics and fitness functions for this world involve computations in binary arithmetic.

The world **small-world** is designed to have relatively simple fitness and physics functions. Both functions have short cycles (modulus 3 and modulus 4). However the composition of the two functions is relatively difficult because the two cycles are out of phase.

## 3.3 Difficulty Measures

To characterize the difficulty of worlds quantitatively, we consider the task of learning either the physics function of the world, or the fitness function of the world, or the function from an environment state to the best available action in that state (i.e., the action that leads to the state with the highest fitness). We would expect the difficulty of these learning tasks to correlate with the unpredictability of the functions to be learned. Intuitively, if similar environments have similar fitness values, or if the actions performed in similar environments lead to environments that are similar, it ought to be relatively easy for the agents to develop adaptive behaviors for those worlds.

To make this intuition precise, we need a way to quantify the "difference" between environment states. Because of its abstractness, our model imposes no particular difference metric. Since the binary representation of the environment state number is used as input to the neural networks in our evolutionary simulations, we use the Hamming distance between the two state numbers (the number of bits where they differ) as a measure of the difference between the states. Other distance metrics, if they were appropriate for a given simulations, could be used in our measures of the difficulty of worlds.

For all worlds:

$$\begin{aligned} E &= \{0, 1, 2, \ldots, 15\} \\ A &= \{0, 1, 2, 3\} \end{aligned}$$

Let $e \in E$ and $a \in A$,

<u>count-flip</u>

$$\begin{aligned} \mathit{physics}\,(e,a) &= e \otimes 2^a \\ \mathit{fitness}\,(e) &= \begin{cases} -1 & \text{if } \sum e_i = 3 \text{ or } 4 \\ 0 & \text{if } \sum e_i = 0 \text{ or } 1 \\ +1 & \text{if } \sum e_i = 2 \end{cases} \end{aligned}$$

$\otimes$ is the bitwise exclusive–or operator;
$e_i$ are the bits of state $e$.

<u>mod-inc</u>

$$\begin{aligned} \mathit{physics}\,(e,a) &= (e+a) \bmod 16 \\ \mathit{fitness}\,(e) &= (e \bmod 3) - 1 \end{aligned}$$

<u>small-world</u>

$$\begin{aligned} \mathit{physics}\,(e,a) &= (e + ((e+a) \bmod 3) + 1) \bmod 16 \\ \mathit{fitness}\,(e) &= \begin{cases} -1 & \text{if } (e \bmod 4) = 1 \\ 0 & \text{if } (e \bmod 4) = 2 \text{ or } 3 \\ +1 & \text{if } (e \bmod 4) = 0 \end{cases} \end{aligned}$$

Figure 1: Specification of the worlds used to model the evolution of behavior.

We define the following three difficulty measures:

The **fitness difficulty** of a world is the root mean square average of the difference between the fitness values of each pair of environments divided by the distance between the environments:

$$\sqrt{\frac{1}{|E|(|E|-1)}\sum_{e_i}\sum_{e_j \neq e_i}\left(\frac{\mathit{fitness}(e_i)-\mathit{fitness}(e_j)}{H[e_i,e_j]}\right)^2} \tag{1}$$

where $|E|$ is the number of different environments, and $H[e_i, e_j]$ is the Hamming distance between the state numbers $e_i$ and $e_j$. In a world with low fitness difficulty, one can often accurately generalize from partial knowledge of the fitness function to values for states that haven't been seen yet.

The **physics difficulty** of an environment state is defined as the average of the distances between all pairs of states reachable from the given state in one action. The physics difficulty of a world is the average of the physics difficulties of all of its states:

$$\frac{1}{|E|}\sum_{e}\left(\frac{1}{|A|(|A|-1)}\sum_{a_i}\sum_{a_j \neq a_i} H[\mathit{physics}(e,a_i),\, \mathit{physics}(e,a_j)]\right) \tag{2}$$

where $|A|$ is the number of actions. In a world with low physics difficulty, most of the actions performed in most states lead to similar environments.

The **action difficulty** of an environment state is the root mean square average of the difference between the fitness values of all of the states reached from that state in one action. The action difficulty of a world is the average of the action difficulties of all its states:

$$\frac{1}{|E|}\sum_{e}\sqrt{\frac{1}{|A|(|A|-1)}\sum_{a_i}\sum_{a_j \neq a_i}[\mathit{fitness}(\mathit{physics}(e,a_i))-\mathit{fitness}(\mathit{physics}(e,a_j))]^2} \tag{3}$$

The action difficulty takes into account both the physics and fitness functions of the world, and measures the degree to which it is important to find the right action in each state. If this value is high, it means that there are lots of states for which one or more of the outcomes has a higher or lower fitness value than the alternatives. Note that this measure does not require a notion of the distance between states.

The difficulty measures for the worlds we used are given in table 2. The worlds count-flip and mod-inc have physics and fitness functions of relatively equal difficulty, but count-flip's action difficulty is much less than that of mod-inc. In small-world, the physics and fitness functions alone are relatively easy, but the action difficulty of small-world is intermediate between that of count-flip and mod-inc. The progress of evolution of agents in these worlds will be influenced by how well the agents' architectures handle these differences.

## 4 Agent Architectures

The agents that populate our simulated worlds each consist of one or more neural networks. All of the networks contain three layers and are fully-connected, and each network uses the logistic activation function for all its nodes. (See Haykin, 1994; chapter 6.) The agents also incorporate mechanisms for determining which action to perform in a given environment state and, in some of the architectures described below, mechanisms for learning.

| count-flip | | | | | |
|---|---|---|---|---|---|
| *env* | *fit* | *a0* | *a1* | *a2* | *a3* |
| 0 | 0 | 1 | 2 | 4 | 8 |
| 1 | 0 | 0 | 3 | 5 | 9 |
| 2 | 0 | 3 | 0 | 6 | 10 |
| 3 | 1 | 2 | 1 | 7 | 11 |
| 4 | 0 | 5 | 6 | 0 | 12 |
| 5 | 1 | 4 | 7 | 1 | 13 |
| 6 | 1 | 7 | 4 | 2 | 14 |
| 7 | -1 | 6 | 5 | 3 | 15 |
| 8 | 0 | 9 | 10 | 12 | 0 |
| 9 | 1 | 8 | 11 | 13 | 1 |
| 10 | 1 | 11 | 8 | 14 | 2 |
| 11 | -1 | 10 | 9 | 15 | 3 |
| 12 | 1 | 13 | 14 | 8 | 4 |
| 13 | -1 | 12 | 15 | 9 | 5 |
| 14 | -1 | 15 | 12 | 10 | 6 |
| 15 | -1 | 14 | 13 | 11 | 7 |

| mod-inc | | | | | |
|---|---|---|---|---|---|
| *env* | *fit* | *a0* | *a1* | *a2* | *a3* |
| 0 | -1 | 0 | 1 | 2 | 3 |
| 1 | 0 | 1 | 2 | 3 | 4 |
| 2 | 1 | 2 | 3 | 4 | 5 |
| 3 | -1 | 3 | 4 | 5 | 6 |
| 4 | 0 | 4 | 5 | 6 | 7 |
| 5 | 1 | 5 | 6 | 7 | 8 |
| 6 | -1 | 6 | 7 | 8 | 9 |
| 7 | 0 | 7 | 8 | 9 | 10 |
| 8 | 1 | 8 | 9 | 10 | 11 |
| 9 | -1 | 9 | 10 | 11 | 12 |
| 10 | 0 | 10 | 11 | 12 | 13 |
| 11 | 1 | 11 | 12 | 13 | 14 |
| 12 | -1 | 12 | 13 | 14 | 15 |
| 13 | 0 | 13 | 14 | 15 | 0 |
| 14 | 1 | 14 | 15 | 0 | 1 |
| 15 | -1 | 15 | 0 | 1 | 2 |

| small-world | | | | | |
|---|---|---|---|---|---|
| *env* | *fit* | *a0* | *a1* | *a2* | *a3* |
| 0 | 1 | 1 | 2 | 3 | 1 |
| 1 | -1 | 3 | 4 | 2 | 3 |
| 2 | 0 | 5 | 3 | 4 | 5 |
| 3 | 0 | 4 | 5 | 6 | 4 |
| 4 | 1 | 6 | 7 | 5 | 6 |
| 5 | -1 | 8 | 6 | 7 | 8 |
| 6 | 0 | 7 | 8 | 9 | 7 |
| 7 | 0 | 9 | 10 | 8 | 9 |
| 8 | 1 | 11 | 9 | 10 | 11 |
| 9 | -1 | 10 | 11 | 12 | 10 |
| 10 | 0 | 12 | 13 | 11 | 12 |
| 11 | 0 | 14 | 12 | 13 | 14 |
| 12 | 1 | 13 | 14 | 15 | 13 |
| 13 | -1 | 15 | 0 | 14 | 15 |
| 14 | 0 | 1 | 15 | 0 | 1 |
| 15 | 0 | 0 | 1 | 2 | 0 |

Table 1: Physics and fitness functions of the worlds used to model the evolution of behavior. The columns headed *env* list the environment states. The columns headed *fit* give the value of the fitness increment for each environment. The columns *a0*, *a1*, *a2* and *a3* indicate the environment that results from performing the corresponding action in the given environment. Thus in the count-flip world, the fitness of state 3 is 1, and performing action 3 in state 3 results in state 11.

| *world* | *fitness* | *physics* | *action* |
|---|---|---|---|
| count-flip | 0.57 | 1.00 | 0.59 |
| mod-inc | 0.60 | 0.98 | 0.92 |
| small-world | 0.42 | 0.76 | 0.73 |

Table 2: Difficulty values for the three worlds.

A neural network's behavior is determined by the numerical values of the connection weights between its nodes. Connection weight values appropriate for a given computation may be computed by a number of different methods, including backpropagation (Rumelhart, et al., 1986) and evolutionary simulations. The latter approach has been explored by Montanta and Davis (1989), Fogel, et al., (1990) and Whitley, et al., (1990). Neural networks are a useful tool for modeling the interaction between evolution and learning because they allow the combination of these two methods for determining connection weight values (Chalmers, 1991; Belew, et al., 1991).

In each time step of its interaction with a world, an agent receives as input a representation of the current environment state as a binary number. It then produces as output a binary representation of one of the possible actions for the world. The physics function of the world determines which environment state occurs next. The value of the fitness function applied to the new environment state is added to a running total. If the agent's architecture includes a learning component, it will modify the weights of one of the agent's networks. This process then repeats in a new environment. After some predetermined number of interactions, the total fitness value accumulated by the agent is divided by the number of interactions, and the resultant average is recorded.

For agents that do not include a learning component (the "innate" architectures below), each agent is exposed to each environment state once; we shall refer to this as one **epoch** of interactions. One epoch is enough to assess the fitness of an agent without a learning component because its behavior will be the same each time it sees the same state. For agents that do include a learning component, the agents interact with each environment state for several hundred epochs. Since the fitness recorded for an agent is the average over all of its actions, the number of epochs the agent experiences doesn't by itself reduce an agent's fitness. However, those agents that must spend time learning to perform well will suffer a penalty compared with those agents that require no learning.

## 4.1 FF-Innate

The **ff-innate** agent architecture, shown in figure 2, contains a single feed-forward neural network. It consists of three layers of nodes, with four nodes in the input layer, six in the hidden layer, and two nodes in the output layer.

A binary representation of the environment state is given to the net as input. Activation values are propagated forward through the network, and the resultant output activation values are used to determine which action to perform. For each output node, if its activation value is greater than 0.5, the corresponding bit in the action's number is 1, otherwise it is 0.[1] The resulting binary number is interpreted as the action.

This architecture does not include a learning component. It is used to explore the evolution of simple "stimulus-response" instinctual action.

## 4.2 FF-Learn

The **ff-learn** architecture, shown in figure 3, incorporates two neural networks. An **action network** takes as input the binary representation of the environment state, and its outputs are used to determine the action to perform. A **reinforcement network** takes as its input the binary representation of the resultant environment state, and computes a signal that is used to modify the weights of the action network. The weights of the reinforcement network, on the other hand, are not changed as the agent interacts with the world. This architecture is similar to the one used in (Ackley and Littman, 1991), except that it has three layers of nodes. It is also similar to the

[1] Recall that the activation values of nodes in neural networks that use the logistic activation function always lie between 0.0 and 1.0.

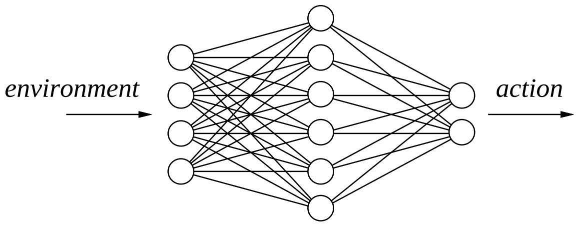


Figure 2: The ff-innate architecture. Bits representing the environment are given to the net as input, and resulting output activations are used to determine the action to perform. The weights of the network are fixed.

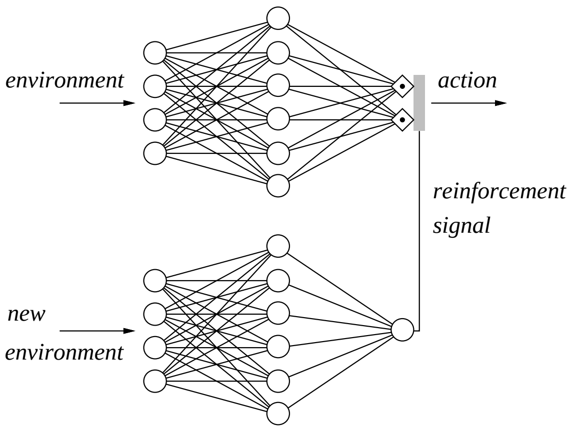


Figure 3: The ff-learn architecture. The action network (top) is used to select actions, based on input from the environment. The output units are shaped like a pair of dice to signify that their values are interpreted stochastically. The reinforcement network (bottom) computes a signal from the resultant environment. This signal is used to direct the modification of the weights of the action network by means of a reinforcement learning algorithm (signified by a fuzzy bar). The weights of the reinforcement network are fixed.

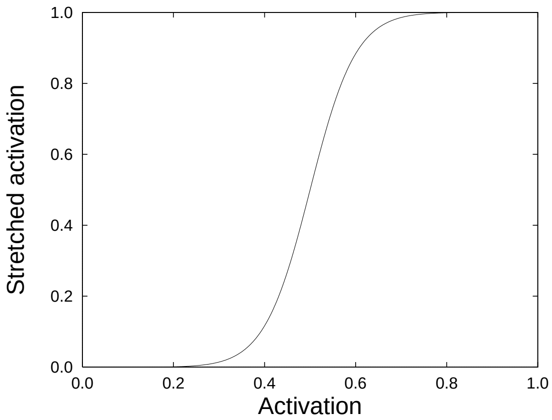


Figure 4: The effect of the stretch factor on activation values. Values of the expression in equation 4 are plotted with $\sigma = 5$.

"adaptive heuristic critic" architecture employed by (Barto, et al., 1990), except that their "critic" is trained directly, whereas our reinforcement network evolves.

The activation values of the output layer of the action network are interpreted stochastically to determine which action to perform. For each output unit, let $\eta_i$ be the value of a random variable uniformly distributed between 0.0 and 1.0. Let $o_i$ be the activation value of the output unit. The corresponding bit in the action chosen is 1 if:

$$\frac{{o_i}^{\sigma}}{{o_i}^{\sigma} + (1 - o_i)^{\sigma}} > \eta_i \tag{4}$$

and is 0 otherwise. The quantity $\sigma$ is a numerical "stretch factor". For our experiments, $\sigma = 5$.

Interpreting the outputs this way, rather than with the simple thresholding function used in the ff-innate architecture, has the consequence that if output activations are near 0.5, the bits generated (and hence the actions performed) are essentially random. This allows the agent to explore alternative actions early in its interactions with the world. As the action network is trained, its output activations will tend to diverge from 0.5 towards either 1.0 or 0.0. The stretch factor accentuates this divergence and makes the agent's behavior more deterministic. For example with $\sigma = 5$, an output activation of 0.6 has a 0.88 chance of being interpreted as a 1.0. A plot of the effect of the stretch factor on activation values is shown in figure 4

After an action is generated, the new environment is fed into the reinforcement network, which computes a single output. This signal is used to direct the modification of the weights of the action network according to the "complementary reinforcement back-propagation" (CRBP) algorithm described by (Ackley and Littman, 1990). CRBP can be used to train networks whose output units are interpreted stochastically, when the only information available is whether the outputs are correct (in whatever sense is appropriate to the domain) or not. If the network's outputs are correct, the weights of the network are modified to increase the likelihood that the network will generate the same output when it sees the same input. If the network's outputs are not correct, the weights are modified to decrease the chance that it will generate the same outputs when it sees the same input, by training the network towards the complement of the incorrect output. The action network's output is taken as correct if the reinforcement network's output on the subsequent environment state is greater than 0.5. We use a learning rate of 1.0 for correct outputs, and 0.1 for incorrect outputs.

Of course "correct" outputs according to the reinforcement network do not necessarily correspond to actions which lead to states of high fitness. The agents never receive any information at all about the fitness values of the states they encounter. The only way that learning could improve an agent's performance, therefore, is if its reinforcement network's outputs are correlated with the fitness values of the environment states as a consequence of evolutionary selection.

## 4.3 RP-Innate

The **rp-innate** architecture, illustrated in figure 5, is another agent architecture for generating innate actions. A single feed-forward **prediction network** takes as input a representation of an environment and an action, and generates a single output value. In a given environment, the action is chosen that, when presented as input to the network along with the environment, results in the maximal activation value of the output node.[2]

The intuition behind this architecture, and the versions of it that incorporate learning, is that the single output value can be thought of as the predicted value of a reinforcement signal (the name "rp" stands for "reinforcement prediction"). The agent is thus choosing the action for which the predicted reinforcement is highest. In this architecture there is no actual reinforcement signal to predict; such a signal appears in the next architecture we describe.

## 4.4 RP-Learn

Learning is added to the reinforcement prediction method of selecting actions in the **rp-learn** architecture, illustrated in figure 6. The prediction network takes as input a representation of the current environment and possible actions. For a given environment, each action $i$ results in an activation value of $o_i$ on the output node. For each action, a random variable $\eta_i$ is taken from a uniform distribution between 0.0 and ${o_i}^\sigma$, and the action for which $\eta_i$ is maximum is chosen. The same stretch factor $\sigma = 5$ used in the ff-learn architecture is used in this one. As in that architecture, the stretch factor enables the agents to explore the domain in their early interactions, while allowing their behavior to become more deterministic as a result of learning.

After the action is performed, a representation of the resultant environment is fed into the reinforcement network, which computes a single output value. This value is then used to train the prediction network's output with the backpropagation algorithm. A learning rate of 1.0 is used. Thus the prediction network is trained to more accurately predict the output of the reinforcement network. To the degree that the reinforcement network's outputs are correlated with the fitness values of environment states, learning will cause the agent to select those actions leading to states with high fitness values.

## 4.5 Model-Learn

The **model-learn** architecture, shown in figure 7, also combines reinforcement prediction and learning. In this architecture, a **model network** takes as inputs both environments and actions. Its output is fed into a reinforcement network which produce a single output value.

[2] As there are only four actions in the worlds we examine, the action is selected by iterating over all of them. If there were too many actions for this method to be practical, the maximal action inputs could be located by a gradient-ascent search. The gradient of a feed-forward network's output as a function of its input activations is computed by setting the error value of the output unit to 1.0, and then applying the backpropagation algorithm to the network, with a learning rate of 0.0. The error value that is thus propagated to an input node equals the gradient of the network's output with respect to that node's value.

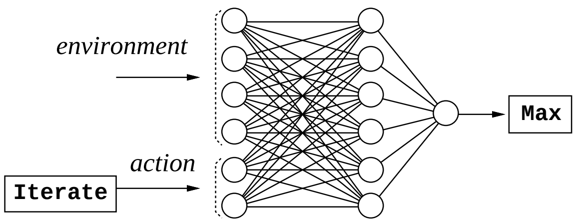


Figure 5: The rp-innate architecture. The network takes as input binary representations of both the environment and potential actions, and computes a single output value. For a given environment, the action chosen is the one for which the output value is maximum. The weights of the network are fixed.

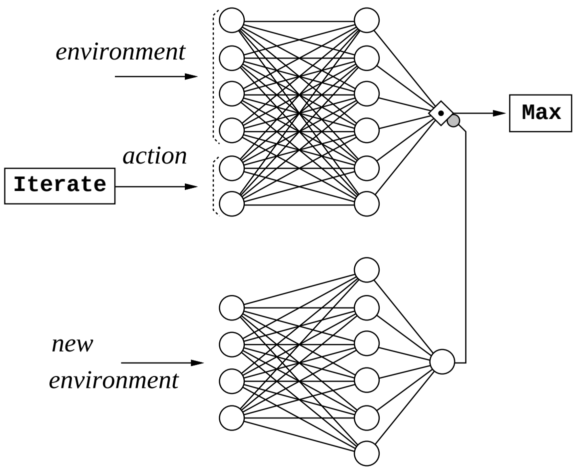


Figure 6: The rp-learn architecture. The prediction network (top) iterates over possible actions in a given environment to locate the one which yields the maximum output of the network. The output unit is interpreted stochastically. This action is then performed, and the resultant environment is presented to the reinforcement network (bottom), whose output is used to train the action network to generate the same value, with the backpropagation algorithm (signified by the small gray circle). The weights of the reinforcement network are fixed.

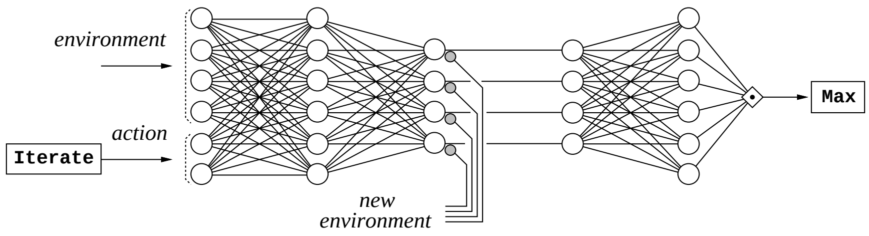


Figure 7: The model-learn architecture. The model network (left), takes representations of the current environment and actions as input and produces as output a representation of an environment. The output values of the model network are fed into the reinforcement network (right), which generates a single output value. Actions are selected by choosing the one for which the output of the reinforcement network (interpreted stochastically) is maximal. The model network is trained by using the environment that results from the action performed. The weights of the reinforcement network are fixed.

Actions for a given environment are selected by iterating over the possible actions. For each action, the corresponding model network outputs are copied directly into the reinforcement network inputs. The action is then selected for which the reinforcement network yields the maximum output, using the same stochastic interpretation procedure as in the rp-learn architecture, again with the stretch factor $\sigma = 5$.

After the action is performed, a representation of the resultant environment is used to train the model network with the backpropagation algorithm, using a learning rate of 1.0. Thus the model network will learn the physics function of the world. If the reinforcement network's output is correlated with the fitness values of environment states, the agent will be able to better select beneficial actions as it improves its predictions of the next states.

We also explore a **model-innate** agent architecture, which is identical to the one pictured in figure 7 except that the weights of both networks are fixed, and the output of the reinforcement network is not interpreted stochastically.

In summary, we investigate six different agent architectures. Three of the architectures contain learning components, and three do not. The architectures also differ in how actions are selected, either as the output of a feed-forward network, or as the result of determining which action input leads to the maximum value of a network's single output. A further distinction involves the learning components. One architecture, ff-learn, uses reinforcement training on its action network, based on its reinforcement network's evaluation of the resultant states. The rp-learn architecture uses backpropagation to train its prediction network to duplicate the output of its reinforcement network. Finally, the model-learn architecture uses backpropagation to train its model network to accurately predict the consequences of its actions.

# 5 Evolutionary Simulations

Each of our simulations is begun by selecting one of the worlds illustrated in table 1, and one of the agent architectures described above. An initial population of 60 agents with that architecture is created. For each of its networks, each agent possesses a vector of numbers equal in length to the total number of connection weights of that network. This set of vectors will be referred to as the agent's **initial weights**. The values of the initial weights of the agents of the first generation of each simulation are generated randomly from a uniform distribution between $+2.5$ and $-2.5$.

In each generation of a simulation, the connection weights of each agent's networks are assigned the values from the agent's initial weight vectors. The agent then interacts with the world for some number of epochs, as described in section 3.

After all of the agents interact with the world, a new population of agents is constructed. The values of the initial weight vectors of the agents in the new population are obtained from the initial weight values of selected members of the current population. For most of the new population, the weights are obtained by selecting an agent from the current generation with a relative probability given by the fitness values the agents achieve in their interactions with the world. The initial weights of the selected agent are copied to the new agent, and are then modified by adding to each value a random number normally distributed with a standard deviation of 0.5. This modification simulates the effects of mutations. In agents with two networks, the initial weights of both networks of the new agent are obtained from the same agent 75% of the time. In the remaining 25% of the cases, two agents are selected, and the new agent gets one vector of initial weights from each agent. This limited form of crossover allows for the possibility that the initial weights of the different networks in an agent could have separable effects. Crossover is not allowed to combine the weights of single networks because a network's performance depends on non-local patterns of its connection weights. To maintain some diversity in the population, 25% of the new population's initial weights are copied exactly from randomly selected members of the current population.

This two-step process of interacting with the world and constructing a new population is then repeated for several hundred generations. Note that although the weights of an agent's networks may change as it interacts with the world, only the values of its initial weight vectors will be transmitted to the next generation.

The specific choices we made for the numerical parameters described above are, for the most part, the result of trial and error. The parameters we used are chosen because they yield a high rate of increase in fitness in all of our simulations. With the exception of the mutation parameter, qualitative aspects of the results described below are unaffected by using different values. If the mutation value is too high (greater than 1.0 or so), offspring of highly fit networks are so unlike their parents that they have little fitness advantage over randomly initialized networks. With the mutation value too low (much less than 0.1), the lack of diversity in the population can lead to premature convergence of the simulations.

Our evolutionary simulations are most similar to the evolution strategies approach of Schwefel (see Bäck and Schwefel, 1993), and the evolutionary programming approach of Fogel (1992), in which inherited information is represented by vectors of real numbers. This representation is more useful for specifying the initial weights of neural networks than that used by the genetic algorithm (Holland, 1975; Goldberg, 1989), in which inherited information is represented by strings of binary digits.

## 6 Results

In this section, we describe the results of our simulations. We first compare runs involving the innate architectures in all three worlds and find that the simulations roughly conform to predictions based on our difficulty measures. Also, the performance of the rp-innate agents is found to be somewhat better than the other two architectures in all three worlds.

We then consider how the learning architectures perform in the different worlds with differing amounts of learning. We find that only in small-world does learning reliably improve the performance of the agents. The simulations involving small-world also show an effect of the sort Baldwin describes, namely that the agents' innate behavior improves as evolution proceeds. Examination of the reinforcement networks in evolved agents shows that their outputs are not perfectly correlated with the fitness of environment states, but are distorted in a useful way.

Finally, we describe simulations of the model-learn architecture, for which learning is also beneficial in small-world. Even though the model networks are trained directly on the physics function of the world they inhabit, the networks acquire a distorted version of the physics function. Rather than degrading the agent's performance, however, the distortions in the model network turn out to be useful in guiding actions.

In figures 8, 9, and 10, results from a number of simulations are presented. Each plot shows the average from ten runs with a specific world and agent architecture. Although some runs lasted 1000 generations, only the first 200 generations are shown, as these best illustrate the differences between the various architectures in the different worlds. Statistics of these runs are summarized in table 3.

## 6.1 Innate Architectures

The top row of figure 8 shows the results of a number of evolutionary simulations of the ff-innate architecture in the three worlds. As can be seen, in all cases the populations are able to improve over the performance of the initial populations as a result of evolution. The populations reached the highest level of average fitness in the count-flip world, consistent with its having the lowest action difficulty of the three worlds. Though mod-inc has a higher action difficulty than small-world, populations in mod-inc reach a slightly higher fitness value than do those in small-world.

The difference between the average fitness at generation 200 of the ff-innate agents in count-flip and mod-inc is significant[3] as is the difference between the average fitness at generation 200 of the ff-innate agents in count-flip and small-world. The difference between the average fitness at generation 200 of the ff-innate agents in mod-inc and small-world is not significant.

The top row of figure 9 shows simulations of the rp-innate architecture in the three worlds. Again the populations do best in count-flip. The differences between the performances between count-flip and mod-inc, and between count-flip and small-world, are both significant. The difference in performance of rp-innate agents in mod-inc and small-world is not significant.

The top row of figure 10 shows shows simulations of the model-innate architecture in the three worlds. The results are consistent with those of the other two innate architectures. The differences between the performances between count-flip and mod-inc, and between count-flip and small-world, are both significant. The difference in performance of model-innate agents in mod-inc and small-world is not significant.

Populations of rp-innate agents perform significantly better than model-innate agents in all three worlds. The performance of rp-innate agents is significantly better than those of ff-innate in mod-inc and small-world, but not in count-flip. The ff-innate agents significantly outperform model-innate agents only in mod-inc.

The rp-innate architecture can do better than the ff-innate architecture because its method of choosing which action to perform is built in. It does not even need to be especially accurate in

[3]In this and the following statistical comparisons, the results from two sets of ten runs are compared. A t–distribution for independent samples with eighteen degrees of freedom is used for a two-tailed comparison of the means of the average fitness values at the last generation of the runs. Unless otherwise indicated, differences whose confidence level $p < .005$ will be referred to as "significant," and differences whose confidence level $p > .02$ will be referred to as "not significant." All of the data used for these comparisons are presented in table 3.

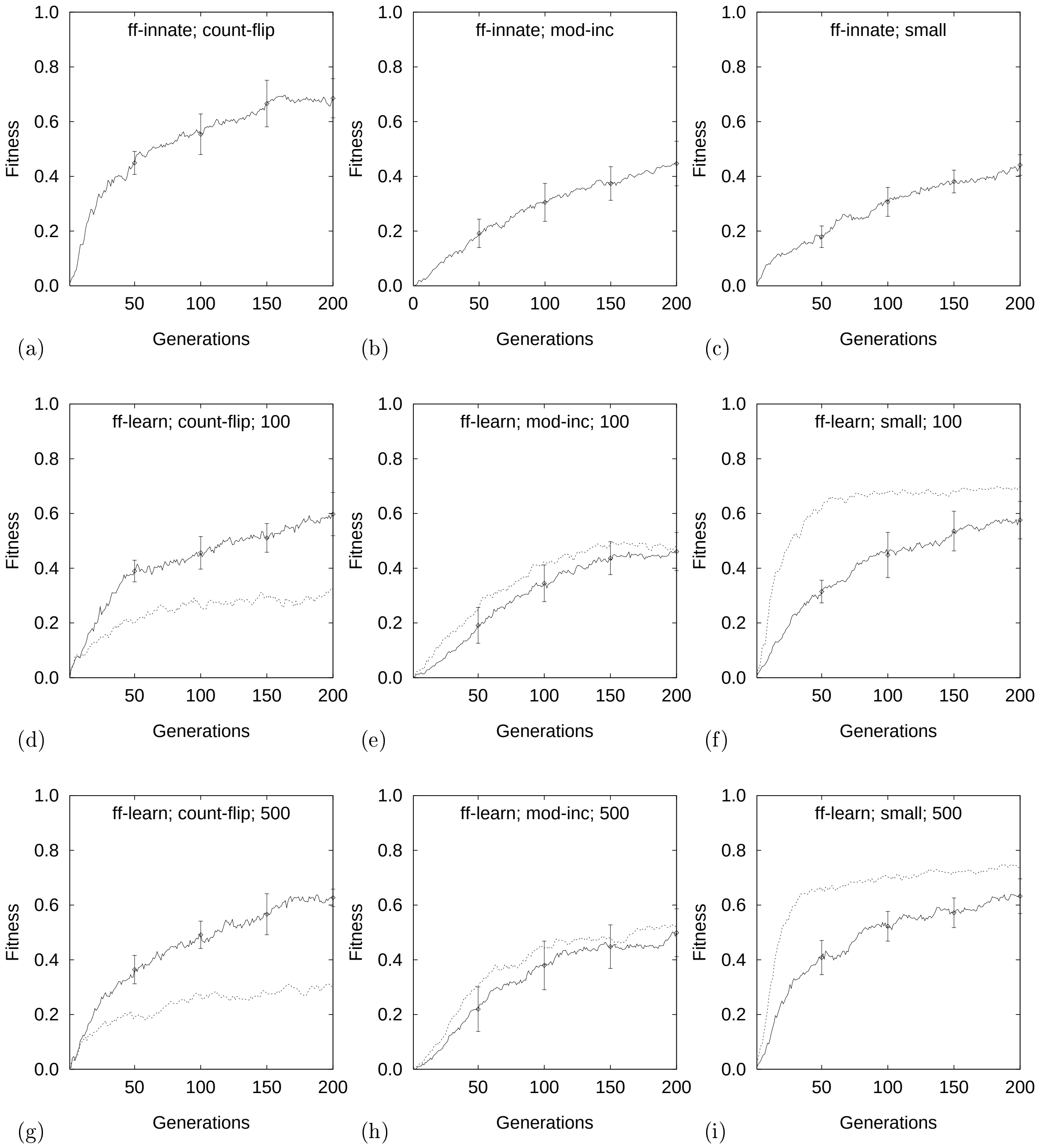


Figure 8: Simulation runs of the ff-innate and ff-learn architectures. The mean value over ten runs of the average fitness of the members of the population at each generation is plotted, with bars indicating the standard deviation. For the simulations of ff-learn agents, the average correlation between the agent's reinforcement network outputs and fitness values of environment states is plotted with dotted lines.

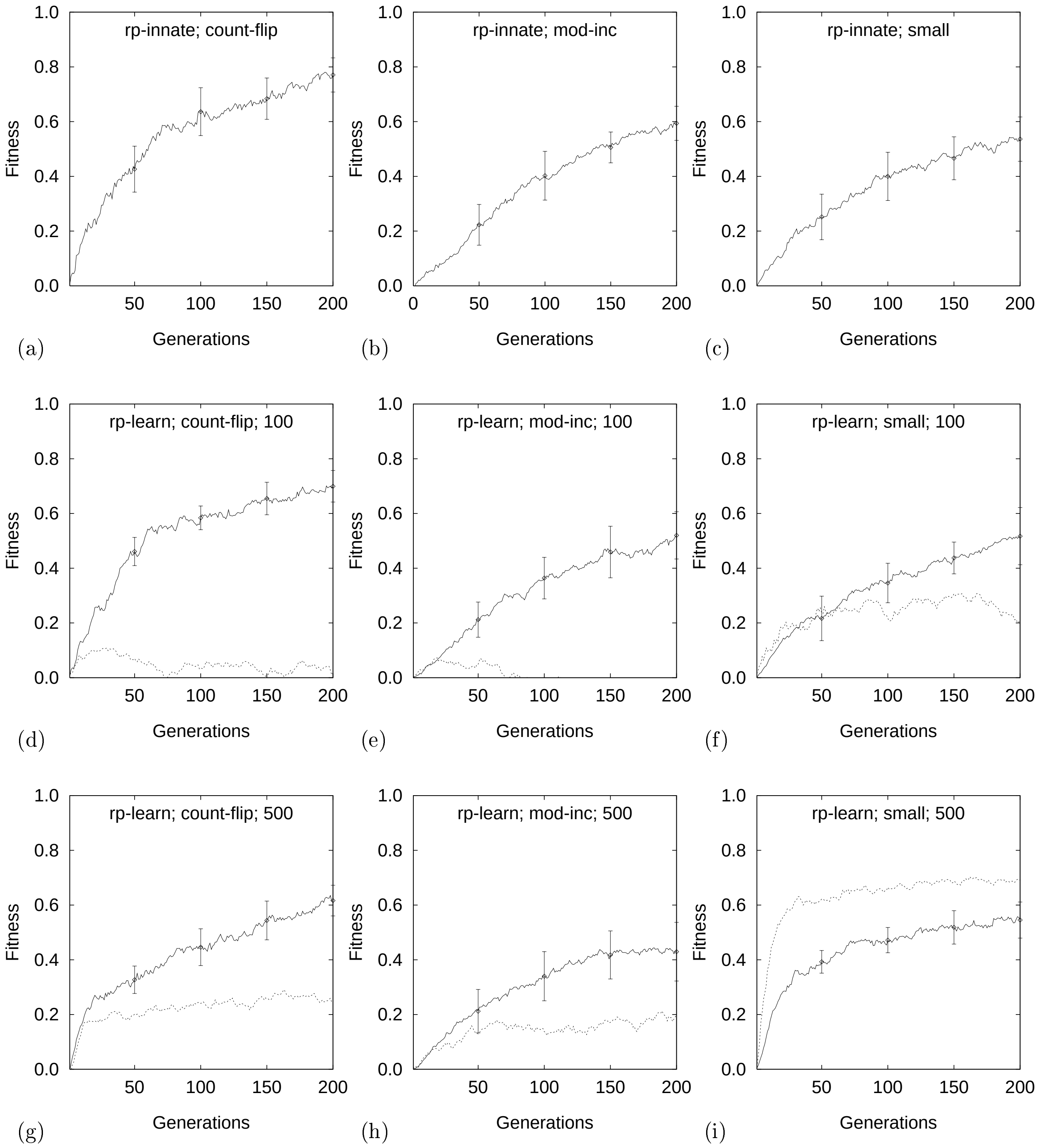


Figure 9: Simulation runs of the rp-innate and rp-learn architectures. The mean value over ten runs of the average fitness of the members of the population at each generation is plotted, with bars indicating the standard deviation. For the simulations of rp-learn agents, the average correlation between the agent's reinforcement network outputs and fitness values of environment states is plotted with dotted lines.

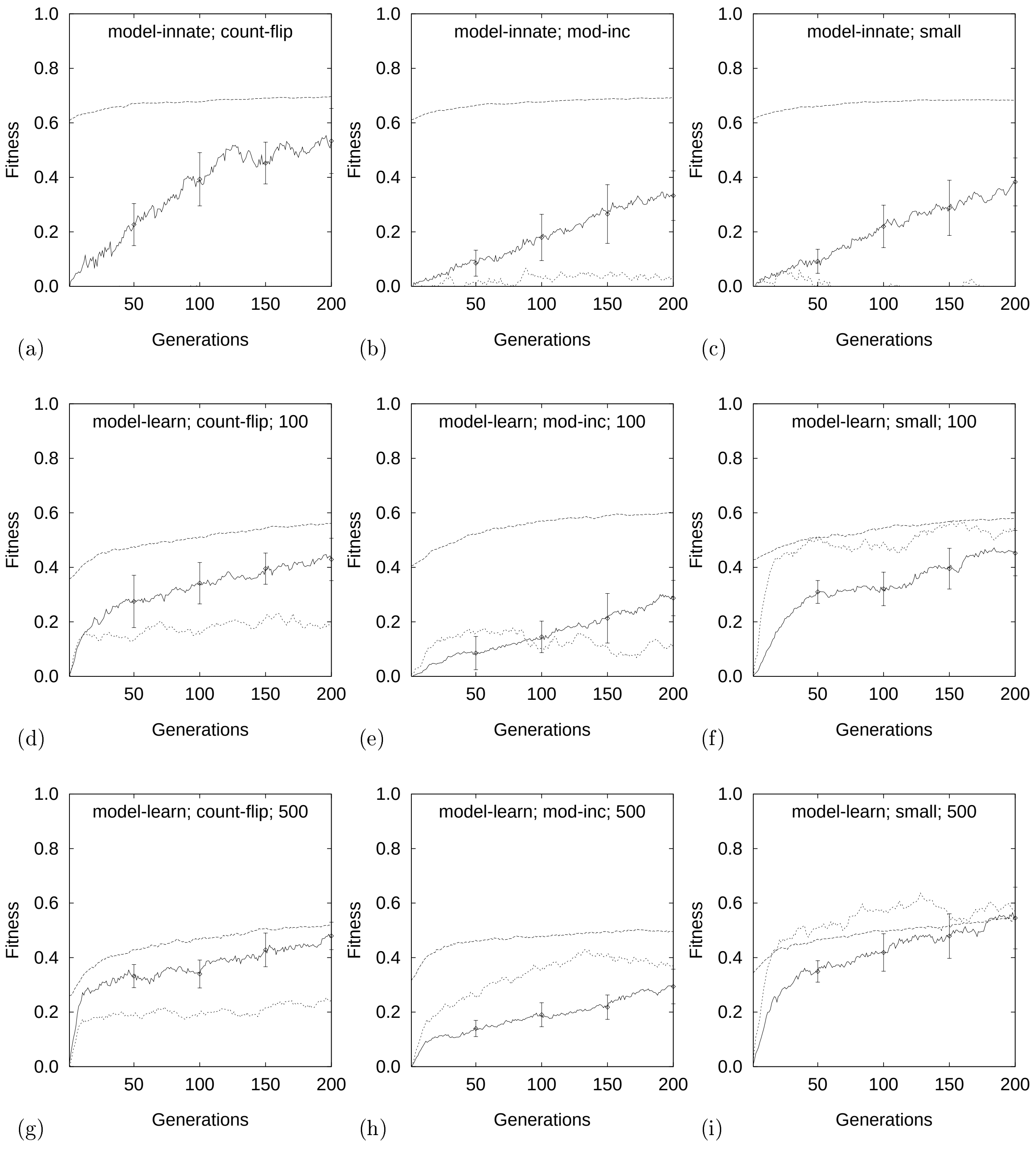


Figure 10: Simulation runs of the model-innate and model-learn architectures. The mean value over ten runs of the average fitness of the members of the population at each generation is plotted with solid lines, with bars indicating the standard deviation. The average correlation between the agent's reinforcement network outputs and fitness values of environment states is plotted with dotted lines. The average error between the agent's model network output for a given state and action, and the bits of the resultant environment state, is plotted with dashed lines.

| ff-innate; count-flip | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .449 | .554 | .666 | .685 |
| *sd* | .042 | .074 | .085 | .071 |

| ff-learn; count-flip; 100 | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .390 | .456 | .511 | .598 |
| *sd* | .040 | .059 | .052 | .079 |

| ff-learn; count-flip; 500 | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .364 | .491 | .566 | .627 |
| *sd* | .052 | .050 | .075 | .032 |

| rp-innate; count-flip | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .426 | .636 | .684 | .771 |
| *sd* | .084 | .087 | .076 | .062 |

| rp-learn; count-flip; 100 | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .461 | .584 | .655 | .699 |
| *sd* | .051 | .043 | .059 | .058 |

| rp-learn; count-flip; 500 | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .327 | .446 | .544 | .616 |
| *sd* | .050 | .067 | .071 | .056 |

| model-innate; count-flip | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .227 | .393 | .453 | .534 |
| *sd* | .077 | .098 | .076 | .119 |

| model-learn; count-flip; 100 | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .275 | .342 | .395 | .429 |
| *sd* | .096 | .076 | .057 | .078 |

| model-learn; count-flip; 500 | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .332 | .340 | .428 | .479 |
| *sd* | .042 | .051 | .062 | .050 |

| ff-innate; mod-inc | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .192 | .305 | .374 | .447 |
| *sd* | .052 | .069 | .061 | .081 |

| ff-learn; mod-inc; 100 | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .191 | .345 | .437 | .461 |
| *sd* | .065 | .067 | .060 | .070 |

| ff-learn; mod-inc; 500 | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .220 | .379 | .448 | .499 |
| *sd* | .082 | .089 | .080 | .087 |

| rp-innate; mod-inc | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .223 | .403 | .506 | .594 |
| *sd* | .074 | .089 | .056 | .062 |

| rp-learn; mod-inc; 100 | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .212 | .364 | .459 | .520 |
| *sd* | .064 | .076 | .094 | .086 |

| rp-learn; mod-inc; 500 | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .212 | .340 | .418 | .430 |
| *sd* | .080 | .090 | .088 | .107 |

| model-innate; mod-inc | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .085 | .179 | .266 | .333 |
| *sd* | .048 | .085 | .108 | .091 |

| model-learn; mod-inc; 100 | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .086 | .145 | .213 | .287 |
| *sd* | .061 | .058 | .091 | .065 |

| model-learn; mod-inc; 500 | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .140 | .190 | .218 | .294 |
| *sd* | .030 | .044 | .045 | .064 |

| ff-innate; small | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .179 | .307 | .381 | .441 |
| *sd* | .039 | .053 | .042 | .037 |

| ff-learn; small; 100 | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .315 | .448 | .536 | .576 |
| *sd* | .041 | .083 | .072 | .069 |

| ff-learn; small; 500 | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .408 | .523 | .572 | .632 |
| *sd* | .063 | .054 | .054 | .063 |

| rp-innate; small | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .252 | .400 | .466 | .536 |
| *sd* | .083 | .088 | .078 | .081 |

| rp-learn; small; 100 | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .217 | .346 | .438 | .517 |
| *sd* | .081 | .072 | .058 | .104 |

| rp-learn; small; 500 | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .393 | .472 | .518 | .545 |
| *sd* | .041 | .046 | .061 | .066 |

| model-innate; small | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .092 | .220 | .288 | .383 |
| *sd* | .044 | .078 | .101 | .088 |

| model-learn; small; 100 | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .310 | .321 | .395 | .452 |
| *sd* | .042 | .061 | .075 | .083 |

| model-learn; small; 500 | 50 | 100 | 150 | 200 |
|---|---|---|---|---|
| *av* | .349 | .419 | .479 | .545 |
| *sd* | .039 | .069 | .082 | .113 |

Table 3: Summary of the simulation runs. Each table contains the mean population fitness (*av*) and its standard deviation (*sd*) at the indicated generation, for ten simulation runs of a specific architecture, world, and number of epochs of interaction.

predicting the fitness consequences of actions, so long as the ordering of their relative fitness values is correct. The ff-innate agent, by contrast, must acquire a mapping from each environment state to the action to perform therein.

The inferiority of model-innate relative to the other two architectures is probably a result of its using an additional network. This addition requires that about 35% more weights be specified. The evolutionary search thus takes place in a significantly higher-dimensional space than that of the other two innate networks. It is therefore slower, even though it uses the same method of action selection as rp-innate.

## 6.2 Learning Architectures

The bottom two rows of figures 8, 9 and 10 show simulations of the three learning architectures, with either 100 or 500 epochs of interactions, in each of the three worlds.

For the ff-learn agents in count-flip, 100 epochs of learning leads to a slight ($p < .05$) decrease in performance relative to the innate architecture. The difference between the innate architecture and the learning version with 500 epochs of interaction is not significant. In mod-inc, learning for either 100 or 500 epochs of interaction does not result in a significant difference between the ff-learn and the ff-innate architectures.

Like the ff-learn agents, the rp-learn agents perform slightly worse than their innate counterparts in count-flip with 100 epochs of interaction, but the difference is significant ($p < .001$) with 500 epochs of interaction. This also happens in mod-inc. In small-world, learning has no significant effect on the performance of rp-learn agents. For the model-learn agents, learning leads to a significant improvement in performance only for small-world with 500 epochs of interaction.

Part of the reason that the learning architectures fail to perform better than the innate architectures has to do with the inherent disadvantage already mentioned: since fitness is assessed after each action, those agents whose innate behaviors are adaptive will have higher fitness than agents that take some time to learn, even if the learning agents ultimately achieve the same level of performance. The fact that the learning architectures perform approximately as well as the innate ones in some of the worlds suggests that learning is having a beneficial effect; however, this effect is not large enough to overcome the disadvantage inherent in the architectures.

Learning is beneficial only in small-world because its fitness difficulty and physics difficulty are low relative to its action difficulty. (See table 2.) Of these three measures, the action difficulty provides the best measure of the overall difficulty of a world, since only the action difficulty incorporates information from both the physics and the fitness functions. For example, the count-flip world's action difficulty of 0.59 is relatively low compared to the other two worlds (0.73 for small-world and 0.92 for mod-inc). The relative simplicity of count-flip explains why most architectures perform better there than in the other two worlds.

However, high action difficulty does not imply correspondingly high physics and fitness difficulties. Judged in terms of action difficulties, the mod-inc world is the most difficult, and the count-flip world is the simplest. However, the fitness and physics difficulties of these two worlds are almost identical (0.57 and 1.0 for count-flip; 0.60 and 0.98 for mod-inc). By contrast, although small-world's action difficulty lies midway between the action difficulties of the count-flip and mod-inc worlds, small-world's fitness and physics difficulties (0.42 and 0.76) are considerably lower than either of the other two worlds.

To evolve adaptive behaviors in mod-inc, an agent must either acquire an accurate reinforcement network, or good innate action strategies, or both. The high fitness difficulty of the world impedes progress on the first solution; the high action difficulty of the world impedes the second.

The relative simplicity of small world's physics and fitness functions explains why learning is beneficial in this world. Learning allows the agent to separate the problem of survival into two subtasks. The innate motivation system "learns" via evolution to evaluate the relative fitness of various environments. This evolved system then allows the agent's learning to focus only upon the physics of the world. For example, in these agents, evolution can find weights for the reinforcement network to mimic the fitness function, and then the prediction or action network can learn to generate the appropriate actions. In small-world, but not in the count-flip or mod-inc world, this decomposition of functions results in an easier task overall, as indicated by the small world's lower physics and fitness difficulties relative to the action difficulty.

The graphs of the simulations of the learning architectures (figures 8 d–i, 9 d–i, and 10 d–i), include plots of the correlation between the output of the reinforcement network, when given as input a representation of an environment state, and the fitness value of that state. We shall refer to this value as the **reinforcement correlation** of an agent.

In the early generations of the simulations, while the reinforcement correlation is low, the agents whose networks initial weight values encode good action strategies perform best. Since the output of the reinforcement network is used to train the other network of the ff-learn and rp-learn agents, it would seem that a high reinforcement correlation value is necessary for learning to benefit an agent. However, as we shall see, learning can be beneficial even with a relatively low reinforcement correlation.

The reinforcement correlation values for the ff-learn agents rise early in the simulations but then level off. In the count-flip simulations of the ff-learn agents (figure 8 d), with 100 epochs of interaction, the reinforcement correlation levels off near 0.25. For the mod-inc world (figure 8 e), the reinforcement correlation value reaches 0.4 late in the simulation. In small-world (figure 8 f), it reaches 0.6 by generation 50 and improves to just over 0.7 by generation 200. The reinforcement correlation values do not rise any higher even when the agents are allowed 500 epochs of interaction in all three worlds.

For the rp-learn agents with 100 epochs of learning, on the other hand, the reinforcement correlations in count-flip and mod-inc barely attain positive values (figure 9 d–e). This suggests that, for these worlds, the rp-learn agents are acquiring innate action strategies and are not benefiting from learning at all. Recall that the innate version of this architecture performed best of the three innate architectures, and performs better than the learning version also. Apparently the superiority of this architecture in acquiring innate action strategies is manifest even when learning is allowed to modify its network's weights.

With 500 epochs of interaction (figure 9 g–i), rp-learn agents acquire a positive reinforcement correlation value in all three worlds, although their performance is worse than that of rp-innate agents in all but small-world. This result supports the suggestion that the primary reason for the improvement in the performance of rp-learn agents is their acquiring innate action strategies. Learning only inhibits this process.

The fact that reinforcement correlation reaches the highest value in small-world for the ff-learn and model-learn agents is consistent with small-world having the lowest fitness difficulty of the three (0.42 for small-world, versus 0.57 for count-flip and 0.60 for mod-inc).

The graphs of the model architectures (figure 10) also include plots of the average root mean square error between an agent's model network output, when it is given a state and action as input, and the environment state that results from performing the given action in the given state. This value shall be referred to as the **model error** for an agent. The model architectures are based on the intuition that adaptive behavior might involve having good models of the world. For model-innate and model-learn agents, this could be accomplished by having the model network accurately model the physics function of the world, and the reinforcement network accurately model the fitness function of the world. Neither happens in the case of model-innate agents. Their reinforcement correlation never rises much above zero. Indeed in the count-flip world the average reinforcement correlation for model-innate agents is negative throughout the simulations. (See figure 10 a–c.)

In all of the simulations of the model-learn architecture, the agents acquire a positive reinforcement correlation. For model-learn agents, however, the reinforcement network is not used for training but is used to select actions. In model-learn agents, the model network is trained directly on the new environment that results from performing an action in a given environment. Since the backpropagation learning algorithm is supposed to reduce error, it is somewhat surprising that in each run of the model-learn agents, this error actually *increases* slightly, even while the performance of the agents improves. This phenomenon will be discussed below.

To explore whether the improvement in performance observed in the simulations of the learning architectures depends on learning, as opposed to the acquisition of innate responses, we performed a set of simulation runs in which the weights of the trained networks are not inherited. Instead, the initial weights of the action networks (in the case of the ff-learn agents) or the prediction networks (in the case of the rp-learn agents) are randomly initialized to values between $-2.5$ and $+2.5$ before the agent interacts with the world for 100 epochs. Thus, the agents cannot acquire innate behaviors and can improve their performance only as a result of learning. The results are shown in figure 11.

For the ff-learn agents, the lack of inherited weights for their action networks resulted in fitness values significantly less than agents whose weights are inherited (figure 8 d–f), even though the reinforcement correlation rises to about the same level as for the agents with inherited action network weights. So while learning is important for these agents, so is the ability to acquire innate adaptive behavior.

For the rp-learn agents, the lack of inherited prediction network weights is even more devastating. Interestingly however, the networks acquire higher reinforcement correlation values than they do when their prediction network weights are inherited (figure 9 d–f). Since they cannot acquire innate actions, these agent must improve their performance by acquiring relatively accurate reinforcement networks.

The fact that learning can improve performance in small-world suggests that one of the effects Baldwin describes might be observed there, namely an improvement in the innate performance of the agents. Each series in figure 12 represents the learning profile of a different generation of a simulation, showing how, at a particular generation, the average fitness of the members of a population changes as a result of learning. A plot of each of the three learning architectures is shown. In all three simulations, the agents interacted with small-world for 100 epochs.

The data indicate that the members of the first generation do not improve at all with learning, which is not surprising, since their reinforcement networks produce essentially random outputs. By generation 20, however, the performance of the agents in all three worlds improves as a result of learning, consistent with the positive reinforcement correlation values for the agents.

In subsequent generations, the agents' performance improves still further with learning, but the untrained performance of the agents also improves. By generation 400 of the simulation of the rp-learn agents, for example, the agents' initial performance is better than the performance of the agents in generation 20 after 100 epochs of learning. What once was mostly learned has now become largely innate, as Baldwin suggested it would.

Suppose, for example, that an agent has a reinforcement network that is fairly well correlated with the fitness functions of its world. If the agent's innate action network weights are already near the values where learning will take them, it will have to spend less time learning than would

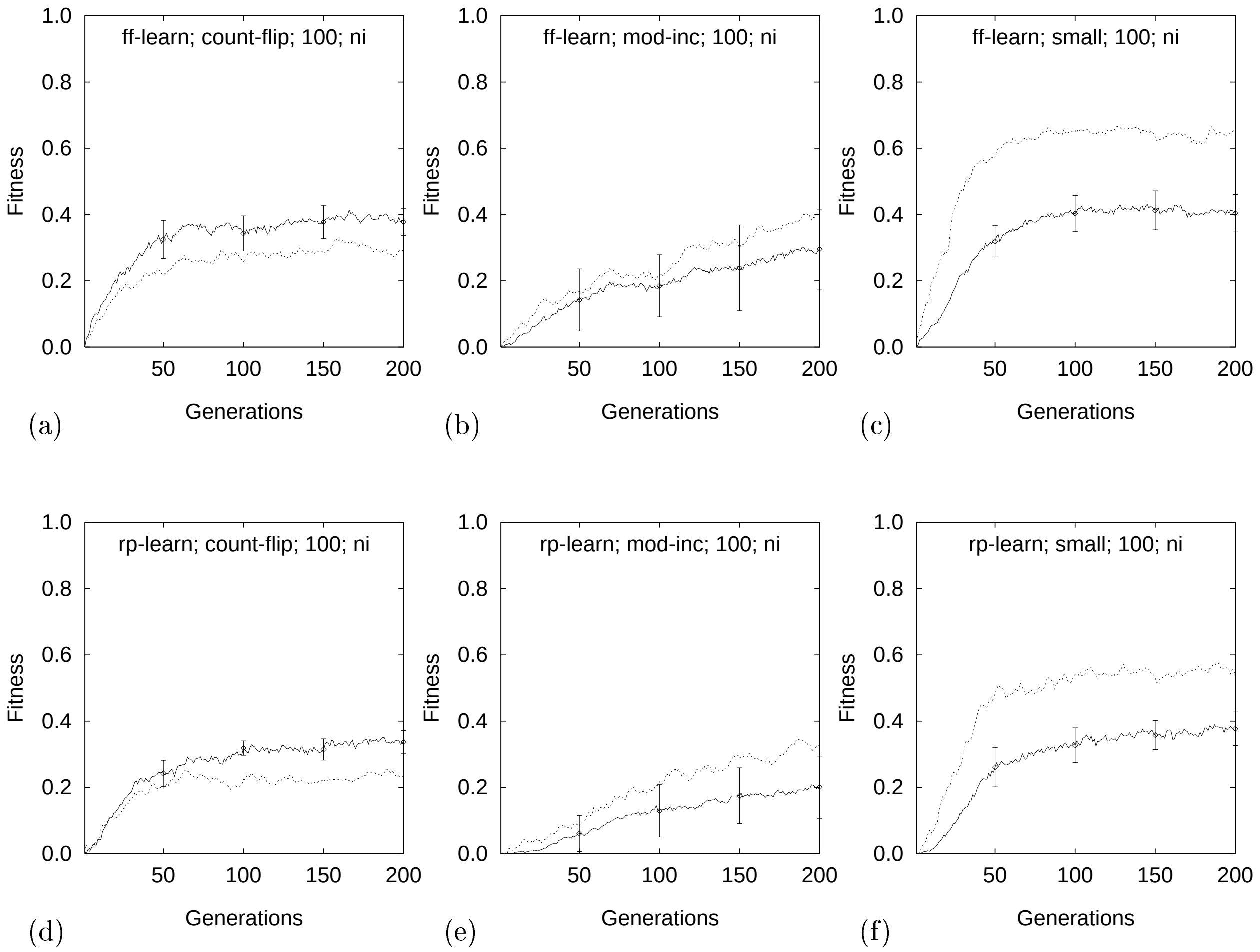


Figure 11: Simulation runs of two of the learning architectures, ff-learn (top), and rp-learn (bottom), in the three worlds, without inherited weights of trained networks. The agents have 100 epochs of interaction with the world.

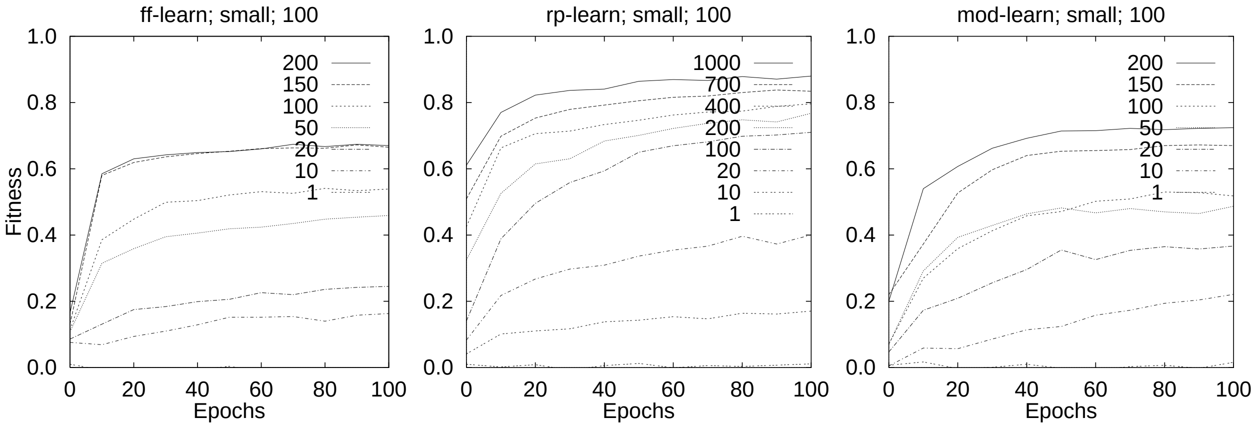


Figure 12: Learning profiles during the evolution of agents in small-world. Each series shows the average fitness of the members of the population after the indicated number of learning rounds. Different series show the performance of subsequent generations. The simulation of the rp-learn agents was continued for 1000 generations.

an agent with randomly initialized action network weights, and will thereby achieve a correspondly higher overall fitness value. So a population of agents whose reinforcement correlation values are equally high will be under selective pressure to improve their innate performance. The acquisition of an innate motivation system (in this case, instantiated in the weights of the agent's reinforcement network), first makes improvement by learning possible, and then reduces its necessity.

## 6.3 Learned Action Strategies

In the following three subsections, we investigate the behavior of individual agents that emerged in simulations of the learning architectures.

### 6.3.1 FF-Learn

Table 4 presents the outputs of the networks of a ff-learn agent from the 200th generation of a simulation in small-world with 100 epochs of interaction, along with a summary of the actions it performs. Recall that the action network is given as input a representation of the current environment state, and the output of the agent's action network is interpreted as a binary number, which is the action performed. (See section 4.2.)

Before learning, the agent's actions aren't very adaptive. It performs the action 0 in most environment states — even in some states where this action leads to states with fitness $-1$. In state 9, though, the naive agent chooses action 2, which leads to state 12, with fitness $+1$.

After learning, the network has acquired a near-optimal set of responses to the world. In state 0 it chooses action 1, leading to state 3, which has fitness 0, but it thereby avoids performing actions 0 or 3, both of which lead to state 1, with fitness $-1$. In state 1 it performs action 1, which leads to state 4, with fitness $+1$. In every environment state in which a state with fitness of $+1$ can be reached, this agent usually chooses an action that does so, except for states 7 and 11, and the agent is very unlikely to choose an action that leads to a state with fitness $-1$.

Due the the stochastic interpretation of the action network output bits, however, the agent will occasionally choose actions that lead to states with fitness values of $-1$. This could happen in state 7, for example, where the output activation of one of the nodes is 0.863, and therefore could be

| *env* | *action output (innate)* | |
|---|---|---|
| 0 | 0.002 | 0.068 |
| 1 | 0.000 | 0.000 |
| 2 | 0.066 | 0.001 |
| 3 | 0.000 | 0.000 |
| 4 | 0.002 | 0.233 |
| 5 | 0.000 | 0.001 |
| 6 | 0.000 | 0.082 |
| 7 | 0.000 | 0.000 |
| 8 | 1.000 | 0.020 |
| 9 | 0.980 | 0.009 |
| 10 | 0.980 | 0.021 |
| 11 | 0.001 | 0.004 |
| 12 | 0.018 | 0.266 |
| 13 | 0.134 | 0.008 |
| 14 | 0.949 | 0.019 |
| 15 | 0.010 | 0.007 |

| *env* | *reinforce output* |
|---|---|
| 0 | 1.000 |
| 1 | 0.004 |
| 2 | 0.992 |
| 3 | 0.001 |
| 4 | 1.000 |
| 5 | 0.000 |
| 6 | 0.145 |
| 7 | 0.000 |
| 8 | 1.000 |
| 9 | 0.261 |
| 10 | 1.000 |
| 11 | 0.001 |
| 12 | 0.984 |
| 13 | 0.005 |
| 14 | 0.745 |
| 15 | 0.000 |

| *env* | *action output (learned)* | |
|---|---|---|
| 0 | 0.184 | 0.932 |
| 1 | 0.000 | 0.922 |
| 2 | 0.918 | 0.052 |
| 3 | 0.001 | 0.103 |
| 4 | 0.440 | 0.960 |
| 5 | 0.000 | 0.159 |
| 6 | 0.087 | 0.909 |
| 7 | 0.000 | 0.863 |
| 8 | 1.000 | 0.023 |
| 9 | 0.923 | 0.008 |
| 10 | 0.991 | 0.914 |
| 11 | 0.000 | 0.041 |
| 12 | 0.105 | 0.888 |
| 13 | 0.000 | 0.963 |
| 14 | 0.978 | 0.054 |
| 15 | 0.002 | 0.093 |

| *env* | *innate action* | *result fitness* | *learned action* | *result fitness* |
|---|---|---|---|---|
| 0 | 0 | -1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 2 | 0 | -1 | 2 | 1 |
| 3 | 0 | 1 | 0 | 1 |
| 4 | 0 | 0 | 1 | 0 |
| 5 | 0 | 1 | 0 | 1 |
| 6 | 0 | 0 | 1 | 1 |
| 7 | 0 | -1 | 1 | 0 |
| 8 | 2 | 0 | 2 | 0 |
| 9 | 2 | 1 | 2 | 1 |
| 10 | 2 | 0 | 3 | 1 |
| 11 | 0 | 0 | 0 | 0 |
| 12 | 0 | -1 | 1 | 0 |
| 13 | 0 | 0 | 1 | 1 |
| 14 | 2 | 1 | 2 | 1 |
| 15 | 0 | 1 | 0 | 1 |

| small-world | | | | | |
|---|---|---|---|---|---|
| *env* | *fit* | *a0* | *a1* | *a2* | *a3* |
| 0 | 1 | 1 | 2 | 3 | 1 |
| 1 | -1 | 3 | 4 | 2 | 3 |
| 2 | 0 | 5 | 3 | 4 | 5 |
| 3 | 0 | 4 | 5 | 6 | 4 |
| 4 | 1 | 6 | 7 | 5 | 6 |
| 5 | -1 | 8 | 6 | 7 | 8 |
| 6 | 0 | 7 | 8 | 9 | 7 |
| 7 | 0 | 9 | 10 | 8 | 9 |
| 8 | 1 | 11 | 9 | 10 | 11 |
| 9 | -1 | 10 | 11 | 12 | 10 |
| 10 | 0 | 12 | 13 | 11 | 12 |
| 11 | 0 | 14 | 12 | 13 | 14 |
| 12 | 1 | 13 | 14 | 15 | 13 |
| 13 | -1 | 15 | 0 | 14 | 15 |
| 14 | 0 | 1 | 15 | 0 | 1 |
| 15 | 0 | 0 | 1 | 2 | 0 |

Table 4: Network outputs and action strategies for a ff-learn agent that has evolved in small-world with 100 epochs of interaction. The table on the top left contains the responses of the agent's action network before it is trained. For each environment state, the networks output values are shown. The table in the middle shows the reinforcement network output for each environment state. The table on the top right shows the responses of the agent's action network after learning. The table on the bottom left summarizes the agent's innate and learned action strategies. For each state, the most likely action performed by the network in that state is shown, followed by the fitness of the environment state that results. The specification of small-world is reproduced from table 1 on the bottom right.

interpreted as 0. (Though with the stretch factor $\sigma = 5$, the chance of this happening is $2 \times 10^{-4}$.) If both bits are interpreted as 0, the agent will end up in state 9, with fitness $-1$.

The training of the action network is controlled by the innate reinforcement network, whose output is also shown in table 4. For each of the environment states that have fitness of $+1$, the reinforcement network output is near 1.0, and for each of the states that have a $-1$ fitness, the reinforcement network output is near 0.0. Recall from figure 8 f, however, that the reinforcement correlation for this type of agent only rises to about 0.7. This is seen in the fact that the reinforcement network has outputs near 1.0 for some environment states whose fitness values are 0, namely states 2, 10 and 14. The reinforcement network thus trains the action network to choose actions that lead to those states.

While a state whose fitness values is 0 doesn't provide as much benefit to an agent as a state with fitness $+1$, it is better for the agent to visit states with 0 fitness than states with $-1$ fitness. The inaccurate reinforcement network prevents the agent from choosing such actions by over-valuing some states. For example, from state 12 no state that has a fitness of $+1$ can be reached. But either action 0 or 3 leads to state 13, with fitness $-1$. The reinforcement network, by responding to state 14 as if it had a high fitness, trains the network to choose action 1 in state 12, thus avoiding the two harmful actions. More examples of how inaccurate reinforcement network outputs can nevertheless lead to adaptive behavior will be seen below.

### 6.3.2 RP-Learn

Table 5 presents the network outputs for an rp-learn agent from generation 200 of a simulation in small-world with 100 epochs of interaction. Recall that rp-learn agents use a network that takes as input a representation of both an environment state and an action. For a given environment, the agent chooses an action to perform by determining which action, when given as input to the prediction network along with the environment state, yields the highest output value. (See section 4.4.)

For this agent, the innate responses are a bit better than for the ff-learn agent examined above. Though it also might choose action 0 in state 0, leading to state 1, with fitness $-1$, this agent tends to choose action 1 in state 1, which leads to state 4, with fitness $+1$. In state 2 it chooses action 2, which also leads to state 4. In state 3 it will choose actions 0 or 3, both of which lead to state 4.

Learning enhances these innate responses and modifies some harmful ones. For example, after learning the network chooses action 1 in state 0, which leads to state 2, with fitness 0. On the other hand, no action from state 0 leads to a state with a fitness of $+1$, and actions 1 and 4, neither of which the trained agent will likely choose, leads to state 1, with fitness $-1$.

For environment states 8 and 9, however, learning has no effect. Both before and after training, the prediction network's output is zero for all actions in these two states. The agent will therefore choose actions randomly in those states, and occasionally will choose action 1 in state 8, leading to state 9, with fitness $-1$.

The rp-learn agent's prediction network is trained on the output of its reinforcement network. The output of this agent's innate reinforcement network is shown in the top center of table 5. Like the ff-learn agent discussed above, this agent's reinforcement network's outputs are not completely consistent with the fitness values of the states. Indeed the output of this agent's reinforcement network and the one shown in table 4 are almost identical. As with the ff-learn agent, the high reinforcement outputs for states 2, 10 and 14 have the effect of training the agent to avoid actions leading to states with fitness $-1$.

Another result of the mismatch between the fitness function and reinforcement network output is that by giving otherwise neutral states high predicted reinforcement, the innate network assures

| | prediction output (innate) | | | |
|---|---|---|---|---|
| *env* | *a0* | *a1* | *a2* | *a3* |
| 0 | 0.001 | 0.001 | 0.000 | 0.000 |
| 1 | 0.000 | 0.758 | 0.000 | 0.000 |
| 2 | 0.302 | 0.000 | 0.996 | 0.139 |
| 3 | 0.188 | 0.001 | 0.000 | 0.149 |
| 4 | 0.056 | 0.001 | 0.157 | 0.101 |
| 5 | 0.004 | 0.031 | 0.000 | 0.131 |
| 6 | 0.252 | 0.004 | 0.144 | 0.141 |
| 7 | 0.002 | 0.000 | 0.999 | 0.000 |
| 8 | 0.000 | 0.000 | 0.000 | 0.000 |
| 9 | 0.000 | 0.000 | 0.000 | 0.000 |
| 10 | 0.999 | 0.182 | 0.000 | 0.054 |
| 11 | 0.000 | 0.182 | 0.000 | 0.000 |
| 12 | 0.497 | 0.214 | 0.000 | 0.000 |
| 13 | 0.000 | 0.108 | 0.000 | 0.000 |
| 14 | 0.267 | 0.141 | 0.999 | 0.144 |
| 15 | 0.999 | 0.002 | 0.000 | 0.999 |

| *env* | *reinforce output* |
|---|---|
| 0 | 1.000 |
| 1 | 0.000 |
| 2 | 1.000 |
| 3 | 0.000 |
| 4 | 1.000 |
| 5 | 0.006 |
| 6 | 0.008 |
| 7 | 0.000 |
| 8 | 0.919 |
| 9 | 0.000 |
| 10 | 0.895 |
| 11 | 0.000 |
| 12 | 0.922 |
| 13 | 0.000 |
| 14 | 0.905 |
| 15 | 0.000 |

| | prediction output (learned) | | | |
|---|---|---|---|---|
| *env* | *a0* | *a1* | *a2* | *a3* |
| 0 | 0.000 | 0.955 | 0.000 | 0.000 |
| 1 | 0.000 | 0.998 | 0.000 | 0.000 |
| 2 | 0.215 | 0.453 | 0.994 | 0.059 |
| 3 | 0.949 | 0.043 | 0.000 | 0.918 |
| 4 | 0.013 | 0.065 | 0.007 | 0.089 |
| 5 | 0.088 | 0.116 | 0.000 | 0.964 |
| 6 | 0.196 | 0.913 | 0.059 | 0.059 |
| 7 | 0.002 | 0.040 | 0.980 | 0.001 |
| 8 | 0.000 | 0.000 | 0.000 | 0.000 |
| 9 | 0.000 | 0.000 | 0.000 | 0.000 |
| 10 | 0.999 | 0.212 | 0.000 | 0.001 |
| 11 | 0.000 | 0.954 | 0.000 | 0.000 |
| 12 | 0.061 | 0.974 | 0.000 | 0.000 |
| 13 | 0.000 | 0.994 | 0.000 | 0.000 |
| 14 | 0.064 | 0.066 | 0.994 | 0.059 |
| 15 | 1.000 | 0.005 | 0.000 | 0.999 |

| *env* | *innate action* | *result fitness* | *learned action* | *result fitness* |
|---|---|---|---|---|
| 0 | 0, 1 | -1, 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 1 | 2 | 1 |
| 3 | 0, 3 | 1, 1 | 0, 3 | 1, 1 |
| 4 | 2 | -1 | 1, 3 | 0, 0 |
| 5 | 3 | 1 | 3 | 1 |
| 6 | 0 | 0 | 1 | 1 |
| 7 | 2 | 1 | 2 | 1 |
| 8 | 0, 1, 2, 3 | 0, -1, 0, 0 | 0, 1, 2, 3 | 0, -1, 0, 0 |
| 9 | 0, 1, 2, 3 | 0, 0, 1, 0 | 0, 1, 2, 3 | 0, 0, 1, 0 |
| 10 | 0 | 1 | 0 | 1 |
| 11 | 1 | 1 | 1 | 1 |
| 12 | 0 | -1 | 1 | 0 |
| 13 | 1 | 1 | 1 | 1 |
| 14 | 2 | 1 | 2 | 1 |
| 15 | 0, 3 | 1, 1 | 0, 3 | 1, 1 |

| small-world | | | | | |
|---|---|---|---|---|---|
| *env* | *fit* | *a0* | *a1* | *a2* | *a3* |
| 0 | 1 | 1 | 2 | 3 | 1 |
| 1 | -1 | 3 | 4 | 2 | 3 |
| 2 | 0 | 5 | 3 | 4 | 5 |
| 3 | 0 | 4 | 5 | 6 | 4 |
| 4 | 1 | 6 | 7 | 5 | 6 |
| 5 | -1 | 8 | 6 | 7 | 8 |
| 6 | 0 | 7 | 8 | 9 | 7 |
| 7 | 0 | 9 | 10 | 8 | 9 |
| 8 | 1 | 11 | 9 | 10 | 11 |
| 9 | -1 | 10 | 11 | 12 | 10 |
| 10 | 0 | 12 | 13 | 11 | 12 |
| 11 | 0 | 14 | 12 | 13 | 14 |
| 12 | 1 | 13 | 14 | 15 | 13 |
| 13 | -1 | 15 | 0 | 14 | 15 |
| 14 | 0 | 1 | 15 | 0 | 1 |
| 15 | 0 | 0 | 1 | 2 | 0 |

Table 5: Network outputs and action strategies for an rp-learn agent that has evolved in small-world with 100 epochs of interaction. The table at top left contains the responses of the agent's prediction network before it is trained. For each environment state and action, the network's output is shown. The table in the middle shows the reinforcement network output for each environment state. The table at top right shows the responses of the agent's prediction network after learning. The table on the bottom left summarizes the agent's action strategies. Actions likely to be performed by the agent are shown, both before and after learning, followed by the fitnesses of the resulting states. The specification of small-world is reproduced from table 1 on the bottom right

that the agent will tend to visit those states, and thereby better learn the consequences of being in them. By limiting the states that the agent learns as it explores the environment, the innate reinforcement network makes the task of learning easier.

### 6.3.3 Model-Learn

Table 6 presents network outputs for a model-learn agent from generation 200 of a simulation in small-world with 100 epochs of interaction. The model network outputs before and after training are shown, as well as the outputs of the agent's reinforcement network. For each environment and action, the correct bits of the resultant environment (as given by the physics function of small-world) are shown, followed by the outputs of the model network when given the environment as input. The upper left table shows the outputs of the model network before learning; the upper right table shows the outputs after 100 epochs of learning. Only the outputs for the first four environments are shown. A summary of the agent's actions for all sixteen states is shown in table 7.

As described in section 4.5, these agents select actions to perform by feeding a representation of the current environment and an action into their model network. The output of the model network is then fed into the agent's reinforcement network, and the action is chosen for which the output of the reinforcement network is highest. After the action is performed the model network is trained directly on the new state.

The model network in table 6 has inherited a very strong bias to make its first output bit 1. Learning doesn't modify this tendency. But for small-world, this bias has the effect of collapsing the sixteen different environments into eight, and, as it turns out for small-world, the fitness of each pair of states thus collapsed is the same. The model network's innate bias thus incorporates a regularity in small-world that it will not have to spend time learning.

Furthermore, the model network, when trained, has distorted its picture of the world even further. For almost all inputs, the network's outputs are wrong, but they are wrong in useful ways. The model's distortions, combined with those of the reinforcement network, enable the agent to select appropriate actions.

In state 0, the trained model network predicts that state 9 will result from action 0, state 10 from action 1, state 9 from action 2, and state 9 from action 3. Of states 9 and 10, the latter has a higher reinforcement value, and hence action 1 is chosen. This leads to state 2, with fitness 0. On the other hand, actions 0 and 3, both leading to state 1, with fitness $-1$, are avoided.

In state 1, the trained model network predicts that state 10 will result from actions 0 and 2, state 8 from action 1, and state 9 from action 3. The reinforcement network's output for state 8 is high, hence the agent chooses action 1, leading to state 4, with fitness $+1$.

In state 2, the trained model network predicts that state 9 will follow action 0, and state 8 will follow actions 1, 2 and 3. It therefore chooses one of these actions, even though action 3 leads to state 5, with fitness $-1$. But before learning the model network predicted that the same state would follow every action, and so the agent might have performed any action in state 2, landing in state 5 about half the time. Furthermore, the output values of the model network in state 2 suggest that with a bit more training (or evolution), the network would predict that state 8 will follow only action 2, which leads to state 4.

In state 3, the trained model network predicts that state 8 will follow actions 0, 2, and 3. Two of these lead to state 4, and the other leads to state 6, with fitness 0. Action 0, leading to state 5, is avoided.

The reinforcement network output for this agent only gives a relatively high value to states whose actual fitness is high in two cases, states 8 and 12. But the model network never shows the reinforcement network states below 8, so the reinforcement network's outputs for such states will

| | | *new* | *model outputs (innate)* | | | |
|---|---|---|---|---|---|---|
| *env* | *act* | *env* | *e0* | *e1* | *e2* | *e3* |
| 0 | 0 | 0001 | 1.000 | 0.000 | 0.199 | 0.983 |
| | 1 | 0010 | 1.000 | 0.775 | 0.987 | 0.002 |
| | 2 | 0011 | 1.000 | 0.001 | 0.183 | 0.996 |
| | 3 | 0001 | 1.000 | 0.984 | 0.176 | 0.875 |
| 1 | 0 | 0011 | 1.000 | 0.000 | 0.995 | 0.005 |
| | 1 | 0100 | 1.000 | 0.032 | 0.998 | 0.118 |
| | 2 | 0010 | 1.000 | 0.000 | 0.485 | 0.915 |
| | 3 | 0011 | 1.000 | 0.000 | 0.689 | 0.872 |
| 2 | 0 | 0101 | 1.000 | 0.000 | 0.044 | 0.884 |
| | 1 | 0011 | 1.000 | 0.011 | 0.082 | 0.879 |
| | 2 | 0100 | 1.000 | 0.000 | 0.125 | 0.995 |
| | 3 | 0101 | 1.000 | 0.017 | 0.037 | 0.888 |
| 3 | 0 | 0100 | 1.000 | 0.000 | 0.801 | 0.059 |
| | 1 | 0101 | 1.000 | 0.034 | 0.887 | 0.567 |
| | 2 | 0110 | 1.000 | 0.000 | 0.934 | 0.000 |
| | 3 | 0100 | 1.000 | 0.000 | 0.081 | 0.998 |

| | | *new* | *model outputs (learned)* | | | |
|---|---|---|---|---|---|---|
| *env* | *act* | *env* | *e0* | *e1* | *e2* | *e3* |
| 0 | 0 | 0001 | 1.000 | 0.000 | 0.008 | 0.954 |
| | 1 | 0010 | 1.000 | 0.001 | 0.920 | 0.000 |
| | 2 | 0011 | 1.000 | 0.000 | 0.013 | 0.980 |
| | 3 | 0001 | 1.000 | 0.003 | 0.050 | 0.806 |
| 1 | 0 | 0011 | 1.000 | 0.000 | 0.947 | 0.002 |
| | 1 | 0100 | 1.000 | 0.000 | 0.125 | 0.038 |
| | 2 | 0010 | 1.000 | 0.000 | 0.900 | 0.000 |
| | 3 | 0011 | 1.000 | 0.000 | 0.025 | 0.560 |
| 2 | 0 | 0101 | 1.000 | 0.000 | 0.045 | 0.695 |
| | 1 | 0011 | 1.000 | 0.000 | 0.104 | 0.464 |
| | 2 | 0100 | 1.000 | 0.000 | 0.011 | 0.014 |
| | 3 | 0101 | 1.000 | 0.000 | 0.098 | 0.453 |
| 3 | 0 | 0100 | 1.000 | 0.000 | 0.033 | 0.008 |
| | 1 | 0101 | 1.000 | 0.000 | 0.007 | 0.962 |
| | 2 | 0110 | 1.000 | 0.000 | 0.432 | 0.000 |
| | 3 | 0100 | 1.000 | 0.000 | 0.009 | 0.013 |

| | *reinforce* |
|---|---|
| *env* | *output* |
| 0 | 0.0041 |
| 1 | 0.0000 |
| 2 | 0.0000 |
| 3 | 0.0000 |
| 4 | 0.0016 |
| 5 | 0.0002 |
| 6 | 0.0030 |
| 7 | 0.0000 |
| 8 | 0.2147 |
| 9 | 0.0001 |
| 10 | 0.0014 |
| 11 | 0.0000 |
| 12 | 0.2235 |
| 13 | 0.0002 |
| 14 | 0.0087 |
| 15 | 0.0000 |

| small-world | | | | | |
|---|---|---|---|---|---|
| *env* | *fit* | *a0* | *a1* | *a2* | *a3* |
| 0 | 1 | 1 | 2 | 3 | 1 |
| 1 | -1 | 3 | 4 | 2 | 3 |
| 2 | 0 | 5 | 3 | 4 | 5 |
| 3 | 0 | 4 | 5 | 6 | 4 |
| 4 | 1 | 6 | 7 | 5 | 6 |
| 5 | -1 | 8 | 6 | 7 | 8 |
| 6 | 0 | 7 | 8 | 9 | 7 |
| 7 | 0 | 9 | 10 | 8 | 9 |
| 8 | 1 | 11 | 9 | 10 | 11 |
| 9 | -1 | 10 | 11 | 12 | 10 |
| 10 | 0 | 12 | 13 | 11 | 12 |
| 11 | 0 | 14 | 12 | 13 | 14 |
| 12 | 1 | 13 | 14 | 15 | 13 |
| 13 | -1 | 15 | 0 | 14 | 15 |
| 14 | 0 | 1 | 15 | 0 | 1 |
| 15 | 0 | 0 | 1 | 2 | 0 |

Table 6: Output of the model network of an agent from the last generation of a simulation of the model-learn architecture in small-world with 100 epochs of interaction. For each environment and each action, the correct bits of the new environment are shown, followed by the output of the model network for that environment and action. The table at the bottom left shows the output of the reinforcement network. The specification of small-world is reproduced on the bottom right.

| *env* | *innate action* | *new state* | *predicted state* | *result fitness* |
|---|---|---|---|---|
| 0 | 1 | 2 | 14 | 0 |
| 1 | 0, 1 | 3, 4 | 10, 10 | 0, 1 |
| 2 | 0, 1, 2, 3 | 5, 3, 4, 5 | 9, 9, 9, 9 | -1, 0, 1, -1 |
| 3 | 0, 2 | 4, 6 | 10, 10 | 1, 0 |
| 4 | 0, 1 | 6, 7 | 10, 10 | 0, 0 |
| 5 | 0, 1, 3 | 8, 6, 8 | 10, 10, 10 | 1, 0, 1 |
| 6 | 0, 1, 2, 3 | 7, 8, 9, 7 | 9, 9, 9, 9 | 0, 1, -1, 0 |
| 7 | 1 | 10 | 10 | 0 |
| 8 | 0, 2 | 11, 10 | 10, 10 | 0, 0 |
| 9 | 0, 1, 2 | 10, 11, 12 | 10, 10, 10 | 0, 0, 1 |
| 10 | 0, 2 | 12, 11 | 10, 10 | 1, 0 |
| 11 | 0, 1, 2, 3 | 14, 12, 13, 14 | 10, 10, 10, 10 | 0, 1, -1, 0 |
| 12 | 1 | 14 | 8 | 0 |
| 13 | 0, 1, 2 | 15, 0, 14 | 10, 10, 10 | 0, 1, 0 |
| 14 | 0, 2 | 1, 0 | 10, 10 | -1, 1 |
| 15 | 0, 2 | 0, 2 | 10, 10 | 1, 0 |

| *env* | *learned action* | *new state* | *predicted state* | *result fitness* |
|---|---|---|---|---|
| 0 | 1 | 2 | 10 | 0 |
| 1 | 1 | 4 | 8 | 1 |
| 2 | 1, 2, 3 | 3, 4, 5 | 8, 8, 8 | 0, 1, -1 |
| 3 | 0, 2, 3 | 4, 6, 4 | 8, 8, 8 | 1, 0, 1 |
| 4 | 0, 1, 3 | 6, 7, 6 | 10, 10, 10 | 0, 0, 0 |
| 5 | 0, 3 | 8, 8 | 8, 8 | 1, 1 |
| 6 | 0, 1, 2, 3 | 7, 8, 9, 7 | 8, 8, 8, 8 | 0, 1, -1, 0 |
| 7 | 2 | 8 | 8 | 1 |
| 8 | 2 | 10 | 10 | 0 |
| 9 | 0, 1, 2, 3 | 10, 11, 12, 10 | 10, 10, 10, 10 | 0, 0, 1, 0 |
| 10 | 3 | 12 | 8 | 1 |
| 11 | 1, 3 | 12, 14 | 8, 8 | 1, 0 |
| 12 | 1 | 14 | 10 | 0 |
| 13 | 1 | 0 | 8 | 1 |
| 14 | 1, 2, 3 | 15, 0, 1 | 8, 8, 8 | 0, 1, -1 |
| 15 | 0, 3 | 0, 0 | 8, 8 | 1, 1 |

Table 7: Action strategies for the model-learn agent whose network outputs are shown in table 6. For each environment state, the actions chosen by the agent are shown, followed by the new states that result from the actions chosen, the states that the agent's model network predicts would follow those actions, and the fitness values of the states that result from the agent's actions.

| | | *new* | *model outputs (learned)* | | | |
|---|---|---|---|---|---|---|
| *env* | *act* | *env* | *e0* | *e1* | *e2* | *e3* |
| 0 | 0 | 0000 | 0.206 | 0.542 | 0.190 | 0.118 |
| | 1 | 0001 | 1.000 | 0.091 | 0.881 | 0.990 |
| | 2 | 0010 | 1.000 | 0.140 | 0.977 | 0.000 |
| | 3 | 0011 | 1.000 | 0.811 | 0.437 | 0.021 |
| 1 | 0 | 0001 | 0.986 | 1.000 | 1.000 | 0.000 |
| | 1 | 0010 | 1.000 | 0.200 | 1.000 | 0.045 |
| | 2 | 0011 | 1.000 | 1.000 | 1.000 | 0.026 |
| | 3 | 0100 | 1.000 | 1.000 | 0.992 | 0.994 |
| 2 | 0 | 0010 | 1.000 | 0.000 | 0.825 | 0.042 |
| | 1 | 0011 | 1.000 | 0.000 | 0.000 | 0.997 |
| | 2 | 0100 | 1.000 | 0.149 | 0.019 | 0.000 |
| | 3 | 0101 | 1.000 | 0.000 | 0.158 | 0.004 |
| 3 | 0 | 0011 | 1.000 | 1.000 | 1.000 | 0.000 |
| | 1 | 0100 | 1.000 | 0.165 | 1.000 | 0.008 |
| | 2 | 0101 | 1.000 | 1.000 | 1.000 | 0.000 |
| | 3 | 0110 | 1.000 | 1.000 | 0.999 | 0.000 |

| *env* | *reinforce output* |
|---|---|
| 0 | 0.0000 |
| 1 | 0.0000 |
| 2 | 0.0000 |
| 3 | 0.0000 |
| 4 | 0.0000 |
| 5 | 0.0000 |
| 6 | 0.0000 |
| 7 | 0.0000 |
| 8 | 0.0439 |
| 9 | 0.0000 |
| 10 | 0.9992 |
| 11 | 0.9988 |
| 12 | 0.0000 |
| 13 | 0.0000 |
| 14 | 0.0000 |
| 15 | 0.0000 |

| mod-inc | | | | | |
|---|---|---|---|---|---|
| *env* | *fit* | *a0* | *a1* | *a2* | *a3* |
| 0 | -1 | 0 | 1 | 2 | 3 |
| 1 | 0 | 1 | 2 | 3 | 4 |
| 2 | 1 | 2 | 3 | 4 | 5 |
| 3 | -1 | 3 | 4 | 5 | 6 |
| 4 | 0 | 4 | 5 | 6 | 7 |
| 5 | 1 | 5 | 6 | 7 | 8 |
| 6 | -1 | 6 | 7 | 8 | 9 |
| 7 | 0 | 7 | 8 | 9 | 10 |
| 8 | 1 | 8 | 9 | 10 | 11 |
| 9 | -1 | 9 | 10 | 11 | 12 |
| 10 | 0 | 10 | 11 | 12 | 13 |
| 11 | 1 | 11 | 12 | 13 | 14 |
| 12 | -1 | 12 | 13 | 14 | 15 |
| 13 | 0 | 13 | 14 | 15 | 0 |
| 14 | 1 | 14 | 15 | 0 | 1 |
| 15 | -1 | 15 | 0 | 1 | 2 |

Table 8: Network outputs for a model-learn agent evolving in mod-inc world. On the left, outputs of the model network after learning. In the center, outputs of the reinforcement network. The specification of mod-inc is reproduced on the right.

never affect the agent's behavior. Furthermore, the output of the model network after learning never has a value much above zero in the second position. Therefore the reinforcement network will never be shown state 12 either.

By collapsing the representations of the states it presents to the reinforcement network, the reinforcement network needs to evolve accurate values for fewer states. Furthermore, the model network has incorporated a simplification of the real physics function, and has also modified it in ways that make performing actions in small-world easier and less risky. The combination of the inaccuracies of the reinforcement network and the model network enable the agent to ignore some aspects of the environment and easily learn the remainder.

Similar systematic distortions are observed in simulations in other worlds. In table 8 the network outputs for a model-learn agent that evolved in the mod-inc world are presented. Again the model network collapses the states of the world by outputting a 1 in the top bit of almost all states. Furthermore, the reinforcement network only has a strong response to two states: state 10, which has an actual fitness value of 0, and state 11, with fitness +1. The model network's outputs are also grossly distorted from the actual physics function. As in the previous example, the combination of the distortions enables the agent to select beneficial actions most of the time.

In the simulation runs of the model-innate agents (figure 10 a–c), the reinforcement correlation never achieves substantially positive values, and the model-error is always high. Thus these networks never acquire accurate models of either the physics or fitness functions of their worlds. A similar effect is observed in simulations of rp-innate agents. Although the output of their prediction networks is used to select actions based on which yields the highest "predicted reinforcement," the correlation between the outputs of these networks and the fitness of environment states never attains positive values. But neither of these results prevented the agents from acquiring adaptive innate behaviors. As with the representations in the networks of the learning agents described above, the distortions introduced by the networks can be put to beneficial use.

Recall from figure 10 d–i, that the average model-error of the agents increases over the simulation run, even though the model network is being trained directly on the world states. The decrease in accuracy is not offset by even 500 epochs of learning.

The model network's distorted representation of the physics function seems to derive from the fact that in early generations of the simulation, none of the reinforcement networks' outputs correlate well with the fitness function. As a result, model networks capable of learning accurately are at no advantage. Instead, those model networks that learn a distorted version of the fitness function, such that the distortions, when fed to an inaccurate reinforcement function, allow the two networks together to find good actions to perform, are better off. Therefore, model networks with tendencies to introduce distortions will be selected. In subsequent generations, the reinforcement networks will always receive input from distorting model networks and will therefore be selected not for their accuracy, but for how well they allow the distorted model network's output to guide actions. Meanwhile, the beneficial effects of the distorted mappings — to avoid dangerous states, to focus learning, and to encode action strategies — decrease the accuracy of the networks still further, while the fitness of the agents increases.

## 7 Discussion

Our approach — of comparing the performance of different architectures, in abstract worlds whose differences can be quantified, with a common simulation and assessment methodology — is intended to provide for the systematic exploration of a wide range of models of the evolution of adaptive behavior. In this work we explore only three very simple worlds; therefore, any generalizations based on our simulations must be provisional. Even so, our results illustrate some of the richness of the interactions between learning and evolution.

The effects Baldwin predicted can be observed in some of our simulations. In simulations involving small-world, populations of agents that incorporate learning evolve more quickly and achieve higher levels of performance than those whose behavior is innate. In addition, the adaptive behavior that must be learned by early generations tends to become increasingly innate. Furthermore, this process can occur while the agents evolve motivation systems to direct the learning of the networks that generate actions.

We also observe, however, situations in which learning offers little or no advantage. Details of the worlds, and the architectures of the agents that evolve in them, can have profound effects on whether and how learning can be beneficial. Learning seems to be most advantageous when, by separating the fitness function of the world from its physics function, the two learning tasks are easier than their composition. This is the case in small-world, but in mod-inc and count-flip the independent tasks are both difficult relative to their composition. Most previous simulations of the Baldwin effect were performed in worlds in which this separation is beneficial.

The distortions introduced by the reinforcement and model networks illustrate a crucial aspect of the way that evolution interacts with learning. Such distortions appear in every simulation we have performed. There seems, in our worlds, to be no adaptive benefit to accurate representations.

In the case of the ff-learn and rp-learn architectures, a distorted reinforcement function has advantages over an accurate one. Such distortions can enable an agent to avoid harmful states, encode action strategies, limit its exploration of the world, and thereby learn a subset of it more accurately.

In the case of the model-learn agents, the distortions are more intriguing. Even though the networks are trained directly on the world they are supposed to model, they inherit strong innate biases to learn inaccurate mappings. As we described above, this is a consequence of the evolutionary histories of the reinforcement and model networks, neither of which ever receives accurate information information from the other, and hence is never under selective pressure to produce accurate information. Once introduced, the distortions are put to beneficial use by evolution, with the networks' biases encoding regularities in the worlds and simplifying the task of learning to behave in it.

We intend to explore more architectures in more complex worlds. We are especially interested in worlds whose physics and fitness functions are stochastic, and which contain state information that is not available to the agents. These agents will therefore have to contain recurrent networks (Jordan, 1988; Elman, 1990) to model their worlds accurately. Such a project could have practical relevance, specifically in those domains where "temporal difference learning" (Barto et al., 1990) methods have been applied. An important problem in such domains is that of determining how an individual output of a network should be reinforced, given that the task involves generating good sequences of outputs. It is possible that for some domains, an evolutionary computation could find the weights of a reinforcement network that could encode a good solution to this problem.

We are also interested in worlds whose environment states and actions are represented as vectors of real numbers, rather than as discrete binary digits, as these are more realistic biologically. We intend to continue comparing our results with, and drawing inspiration from, what is known about the interactions between evolution, learning, and the mechanisms of action selection in animals (Kupfermann, et al, 1970; Epstein, 1982; Thompson, 1986; Toates, 1986; Colgan, 1989; McFarland and Bosner, 1993; Montague, et al, 1995).

# 8 Conclusions

Baldwin's analysis of the interactions between evolution and learning relied on an assumed "fact of physiology" that animals tend to repeat actions that are beneficial and avoid those that are harmful. But this tendency requires that animals can tell the difference. In this work we have examined this assumption more closely. We found that a motivation system can indeed coevolve with the learning of the behaviors it evaluates.

However, the evolved motivation systems are not very accurate. The synergy between learning and evolution introduces systematic distortions into the representations of the world that the motivation systems compute. These distortions are often highly beneficial, as they can incorporate regularities in the world, simplify the learning task, and encode strategies for adaptive behavior.

# Acknowledgments

The authors thank the reviewers of this paper for their thoughtful and valuable suggestions and comments. William Grundy is funded by the National Defense Science and Engineering Graduate Fellowship Program.

# References

D. H. Ackley and M. L. Littman. Generalization and scaling in reinforcement learning. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 550–557, San Mateo, CA, 1990. Morgan Kaufmann.

D. H. Ackley and M. L. Littman. Interactions between learning and evolution. In C. G. Langton, C. Taylor, J.D. Farmer, and S. Rasmussen, editors, *Artificial Life II*, volume X of *Santa Fe*

*Institute Studies in the Sciences of Complexity*, pages 487–507, Reading, MA, 1992. Addison Wesley.

Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.

James Mark Baldwin. A new factor in evolution. *American Naturalist*, 30:441–451, 1896.

Andrew G. Barto, Richard S. Sutton, and Christopher J. C. H. Watkins. Learning and sequential decision making. In Michael Gabriel and John Moore, editors, *Learning and Computational Neuroscience: Foundations of Adaptive Networks*, chapter 13, pages 539–602. Bradford Books/MIT Press, 1990.

John Batali. Innate biases and critical periods: Combining evolution and learning in the acquisition of syntax. In Rodney Brooks and Pattie Maes, editors, *Proceedings of the Fourth Artificial Life Workshop*, pages 160–171, Cambridge, MA, 1994. The MIT Press.

R. K. Belew. When both individuals and populations search: Adding simple learning to the genetic algorithm. In J. D. Shaefer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*. Morgan Kaufman, 1989.

R. K. Belew, J. McInerney, and N. N. Schraudolph. Evolving networks: Using the genetic algorithm with connectionist learning. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life II: SFI Studies in the Sciences of Complexity, Volume X*, pages 511–547, 1991.

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

Patrick Colgan. *Animal Motivation*. Chapman and Hall, 1989.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

A. N. Epstein. Instinct and motivation as explanation for complex behaviour. In D. W. Pfaff, editor, *The Physiological Mechanisms of Motivation*, pages 25–28. Springer, 1982.

D. B. Fogel. An analysis of evolutionary programming. In D. B. Fogel and J. W. Atmar, editors, *Proceedings of the First Annual Conference on Evolutionary Programming*, 1992.

D. B. Fogel, L. J. Fogel, and V.W. Porto. Evolving neural networks. *Biol. Cybern.*, 63:487–493, 1990.

D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Mass., 1989.

D. W. Hamlyn. *Aristotle's De Anima*. Clarendon/Oxford University Press, 1968.

Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Macmillan, New York, 1994.

G. Hinton and S. J. Nowlan. How learning can guide evolution. *Complex Systems*, pages 495–501, 1987.

J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

C. L. Hull. *A Behaviour System*. Yale University Press, 1952.

Michael Jordan. Serial order: a parallel distributed processing approach. Technical Report ICS Report No. 8604, Institute for Cognitive Science; University of California at San Diego, 1986.

I. Kupfermann, V. Castellucci, H. Pinsker, and E. Kandel. Neuronal correlates of habituation and dishabituation of the gill withdrawal reflex in *aplysia*. *Science*, 14(167):1743–1745, 1970.

Michael Littman. Simulations combining evolution and learning. In R. K. Belew and M. Mitchell, editors, *Adaptive Individuals in Evolving Populations*. Addison-Wesley, 1996.

David McFarland and Thomas Bösner. *Intelligent Behavior in Animals and Robots*. MIT Press, Cambridge, MA, 1993.

P. R. Montague, R. Dayan, C. Person, and T. Sejnowsky. Bee foraging in uncertain environments using predictive Hebbian learning. *Nature*, 377:725–728, October 1995.

D. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. In *Proceedings of the 11th IJCAI*, 1989.

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

Daniel N. Robinson. *Aristotle's Psychology*. Columbia University Press, New York, 1989.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, volume 1. MIT Press, Cambridge, MA, 1986.

R. F. Thompson. The neurobiology of learning and memory. *Science*, 233:941–947, 1986.

Edward L. Thorndike. *Animal Intelligence*. Macmillan, New York, 1898.

Frederick Toates. *Motivational Systems*. Cambridge University Press, 1986.

P. M. Todd and G. F. Miller. Exploring adaptive agency II: Simulating the evolution of associative learning. In J. A. Meyer and S. W. Wilson, editors, *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 306–315, Cambridge, MA, 1991. Bradford Books/MIT Press.

D. Whitley, T. Starkweather, and C. Bogart. Genetic algorithms and neural networks — optimizing connections and connectivity. *Parallel Computing*, 14(3): 347–361, August 1990.