Chapter 4

LEARNING TO CONTROL A FINITE STATE AUTOMATON

In this chapter we discuss a control acquisition architecture that uses a model to determine a minimal length sequence of actions to reach a particular goal state. The architecture consists of a *model* and an *adaptive critic*. The basic architecture is shown in Figure 4.1 and will be called the *control network*. The model maps its current state and the current action to the predicted next state, and the adaptive critic maps the model's next state to the expected time to reach the goal state. The model's state consists of both predicted sensations and hidden state information, corresponding to the output and hidden units of SLUG.

At each time step the control action is chosen which minimizes the output of the adaptive critic. This is done by using the control network to evaluate, at each time step, each possible control action. The control network computes a prediction of the time to reach the goal state if that particular action were taken in the current state. The control network performs this computation in two parts: first, the model computes the state q predicted to result from taking that particular action; second, the adaptive critic computes the prediction of the time to reach the goal state from state q. The action which results in the minimum prediction of time to reach the goal state is then performed.

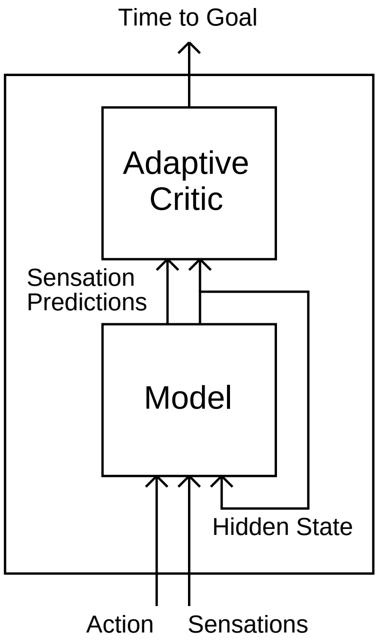

Figure 4.1. Connectionist Control Acquisition Architecture.

4.1 Training

In our experiments, our training procedure is divided into two phases. First, the model is trained during an exploration phase using the unconstrained version of SLUG. The weights of SLUG are then fixed. In the second phase, the control network is trained to produce optimal trajectories from arbitrary initial states to a particular goal state. This phase involves a series of trials which begin and end in the goal state. At the beginning of each trial, the robot explores for a fixed number of steps by taking actions selected randomly from a uniform distribution. The number of such steps is chosen to be the number of states in the environment. This number was chosen so that there is a nonzero probability of the exploration phase ending in any particular environmental state. This places the robot in a random state, which we regard as the initial state for the remainder of the trial. [1] During the rest of the trial, the

[1]There are two motivations for this wandering procedure. First, in a learning situation with minimal supervision, the robot needs to get into the initial state on its own; it cannot depend on

robot performs the actions that produce minimal predictions of the time to reach the goal state according to its current model and adaptive critic. In order to encourage exploration during control trials, the robot sometimes (with probability 0.25 at each time step in all the simulations [2]) selects alternative actions instead of selecting the action that is best according to the current predictions. This exploration strategy is not optimal, but provides the necessary exposure to the state space. Sutton [94] and Watkins [97] present more sophisticated techniques for exploration. Throughout the entire control acquisition phase—during wandering and during the control trials—the model constantly updates itself to reflect the current state of the environment.

The adaptive critic is trained using a *temporal difference (TD)* method developed by Sutton [92, 93]. A system using this method learns to predict by maintaining consistency across successive predictions and by forcing the prediction to agree with any available training signals. In the case of predicting the time to reach the goal state, the adaptive critic is trained to make successive predictions decrease by one and to make the prediction at the goal state zero. The robot receives a reinforcement signal that indicates when it is located at the goal; the reinforcement signal is 1 when the robot is located at the goal and 0 otherwise. Finally, the adaptive critic is trained only when the robot actually selects the action that currently looks best and not when the robot takes exploratory steps (see Watkins [97] for justification).

The adaptive critic is implemented as either a one-layer or two-layer network with a single linear output unit. The hidden layer is unnecessary for the little prince and $n \times n$ checkerboard environments. The adaptive critic's optional hidden layer consists of logistic units with outputs ranging from -1 to $+1$.

---

a supervisor to place it in a random initial state. Second, the model needs to reflect the current environmental state. If the robot is placed in a random state, substantial effort would be required to identify the environmental state. However, by starting at the goal state and wandering from there, the robot can maintain continual knowledge of the environmental state.

[2] After trying a few different probabilities, 0.25 was found to produce the shortest learning times.

Table 4.1. The Control Acquisition Architecture was Applied to these Environments with their Respective Goal States.

Environment	Goal State
Little Prince	Robot on the north location facing east
Car Radio	Playing Classical, Rock in preset X, and News in preset Y
3-Bit Shift Register	Bit pattern `101`
4-Bit Shift Register	Bit pattern `1010`
4×4 Checkerboard	Square `(1,1)`
6×6 Checkerboard	Square `(2,2)`
8×8 Checkerboard	Square `(3,3)`

We use the 3-bit shift register environment to illustrate the control acquisition architecture. Initially, an unconstrained SLUG model of this environment was constructed with the techniques described in Chapter 2.4. Next, using the units of SLUG as inputs to an adaptive critic network, the critic was trained to indicate the minimal number of time steps to achieve the bit pattern `101` from any possible initial bit pattern. Once trained, the control network implicitly defines a *control law* that specifies the shortest (or equivalently minimal time) action sequence required to achieve the goal state.

4.2 Simulation Results

The control acquisition architecture was applied to the environments listed in Table 4.1. In all the environments, goal states were chosen to be indistinguishable from other states based only on the sensations. For example, to be considered in the goal state in the car radio environment, the robot must have the rock station stored in preset X, the news station stored in preset Y, and must be listening to the classical station. In the $n \times n$ checkerboard environment, the goal location is a particular square that does not contain a landmark and that is located in the middle of the grid. The coordinates given in Table 4.1 are relative to the bottom left square `(0,0)`.

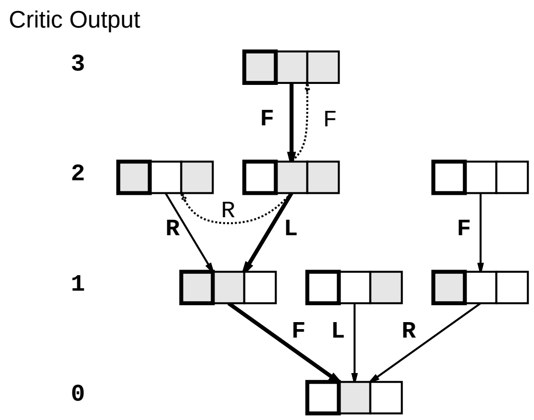

Figure 4.2. Summary of the Control Law and Evaluation Function for the 3-Bit Shift Register Environment. Each state is depicted as a 3-bit shift register where grey denotes a value of 0 in a particular cell, white a value of 1. Arrows indicate some of the possible transitions from one state to another. Solid arrows correspond to the optimal actions; dashed arrows correspond to suboptimal actions.

Figure 4.2 summarizes the resulting control law and evaluation function learned for the 3-bit shift register environment. All states of the shift register are shown, including the goal state, `101`, which is at the bottom of the figure. Each state is depicted as a shift register with the corresponding bit settings, where grey denotes a value of zero, and white denotes a value of one. The states are depicted in decreasing order from top to bottom in terms of the output of the adaptive critic for that state (i.e., expected minimum number of time steps needed to reach the goal state). This figure summarizes the implicit control law by showing the optimal transitions from state to state. For example, suppose the robot is in state `100` and is going to choose the best next action. The robot could choose action `L` resulting in state `001` with value 1, or action `R` resulting in state `010` with value 2, or action `F` resulting in state `000` with value 3. The best action available to the robot, according to the model and adaptive critic, is `L` which is shown with the solid line; the suboptimal actions (`F` and `R`) are shown by dotted lines in Figure 4.2.

Table 4.2. An Example State/Action Sequence for the Car Radio Environment.

State			action
current	preset X	preset Y	
classical	classical	classical	scan left
news	classical	classical	store in preset Y
news	classical	news	scan left
rock	classical	news	store in preset X
rock	rock	news	scan left
classical	rock	news	—

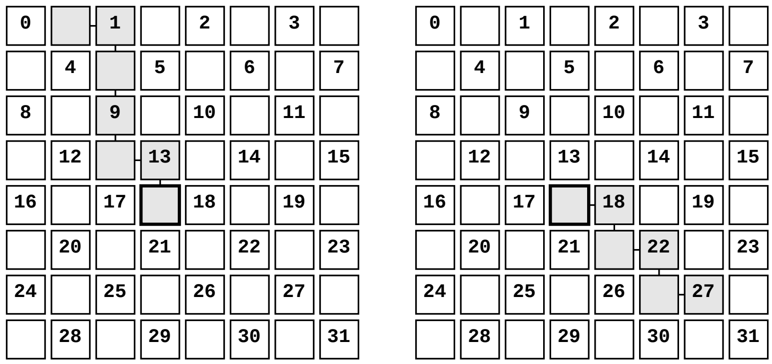

Figure 4.3. Two Example Trajectories for the 8×8 Checkerboard Environment. Each trajectory is shown as a sequence of shaded squares connected by line segments and ending in the goal square.

During non-exploratory steps, the robot chooses the estimated optimal action given the current state. For example, if the shift register starts in state `000`, the optimal action sequence is `FLF`. This trajectory is depicted in Figure 4.2 as a sequence of bold arrows connecting state `000` to state `101` via states `100` and `001`.

As a further demonstration of the control network, we describe the behavior of the network on the car radio and 8×8 checkerboard environments after training. In the car radio environment, the robot learns to store the proper stations in the presets and then to play classical music. One example sequence of state/action pairs is shown in Table 4.2.

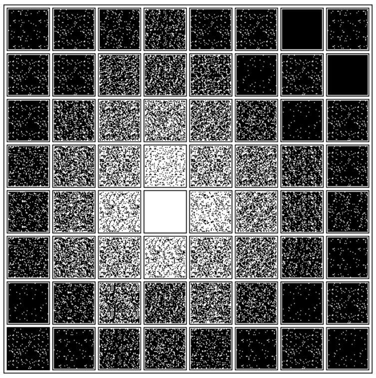

Figure 4.4. The Output of the Adaptive Critic at Each of the Squares in the 8×8 Checkerboard Environment. The evaluations are portrayed with white being the smallest time to goal and black being the largest time to goal.

In the 8×8 checkerboard environment, the goal state was chosen to be square `(3,3)`. The resulting controller is able to choose optimal actions from each square in the grid, i.e., actions that move it one step closer to the goal position. Figure 4.3 shows two trajectories for this environment, and Figure 4.4 shows the relative output of the adaptive critic in all the squares. The evaluations decrease steadily to zero at the goal square `(3,3)`.

To evaluate the control acquisition architecture, we examined its performance as measured by the total number of actions that must be executed before the system produces optimal goal-seeking behavior. The total number of actions include wandering actions, exploration actions, as well as actual control actions during which learning occurs. The system was considered to have learned a task by a given time step if the optimal control actions are taken for at least the next 2500 actions. In order to judge whether a control action was optimal, a table of the minimum number of time steps needed to reach home for each state was constructed prior to training.[3]

[3] One might wonder why we need an adaptive critic if we can construct a table. First, an adaptive critic permits the learning of minimal length sequences of actions in nonstationary environments. Secondly, an adaptive critic permits the learning of minimal length sequences of actions for larger environments for which constructing a table is impractical.

Table 4.3. Performance of the Control Acquisition Architecture.

Environment	# of Env. States	# of Hidden Units	Learning Rate	Median Number Needed to Learn Optimal Behavior		
				Actions		Trials
				All	Control	
Little Prince	4	0	0.100	20	8	2
3-Bit Shift Register	8	8	0.050	4443	778	373
4-Bit Shift Register	16	16	0.020	7662	1937	282
Car Radio	27	16	0.010	12512	2199	355
4×4 Checkerboard	16	0	0.001	6061	1114	262
6×6 Checkerboard	36	0	0.001	57915	5692	1327
8×8 Checkerboard	64	0	0.001	106507	8777	1446

An action that changes the environmental state from x to y was considered optimal if the minimum number of time steps needed to reach home from state y is one less than the minimum number needed from state x. For each environment, several runs were made to determine the learning rate for the adaptive critic that yielded the best performance. Because results varied from one run to another due to the random initial weights, we report the median performance over 25 replications of each simulation. On all replications, the control network was able to converge to an optimal solution.

Table 4.3 reports the performance of the control acquisition architecture for the environments listed in Table 4.1. Table 4.3 lists the number of environmental states, the number of logistic hidden units contained in the adaptive critic network (where 0 in this column means that a hidden layer was unnecessary), and the learning rate used for training the adaptive critic. Table 4.3 also reports the median total number of actions, number of control actions, and number of trials needed to learn to produce optimal behavior. Because the control network was only trained during control actions, the number of control actions equals the number of weight updates. The results of Table 4.3 do not include the time needed to train the model.

Because of the large number of wandering and exploratory actions, the ratio of the number of control actions to the total number of actions is small. More sophisticated wandering and exploration procedures would substantially increase this ratio. Figure 4.5 shows some learning curves for the 4-bit shift register environment. The curves demonstrate that the system rapidly learns the majority of the control law and then takes significantly longer to learn the residual behavior.

4.3 Relation to Other Work

In this section only a small sample of the connectionist control literature is covered. Consult Barto [6] and Narendra [61] for more comprehensive overviews of this subject. In this section, the focus is on the methods utilized in our control acquisition architecture.

Supervised learning requires an informative teacher that can provide appropriate target values. Unfortunately, in control tasks the environment can rarely give the desired control signals. Jordan and Rumelhart [37] describe a technique that permits supervised learning to be applied in these cases. The technique, called *forward modeling*, involves building a model of the environment, and it allows the controller to be optimized in terms of a performance criterion defined in distal coordinates.

Consider the problem of learning the inverse kinematics of a multi-joint robot arm with excess degrees of freedom (as described in Jordan and Rumelhart [37]). The inverse kinematics is a mapping from a spatial position vector $\boldsymbol{x}$ to the joint angles $\boldsymbol{\theta}$ necessary to move the endpoint of the arm to that position. Learning the inverse kinematics directly is difficult when there are excess degrees of freedom because there are many joint angles that result in the same position. Standard function approximation techniques would tend to learn the average of the joint angles for a given position. Unfortunately, the average might not be a valid solution. As an additional difficulty, the teaching signal provides desired positions and not desired joint angles. Therefore, the learning algorithm must somehow convert position errors into joint angle errors.

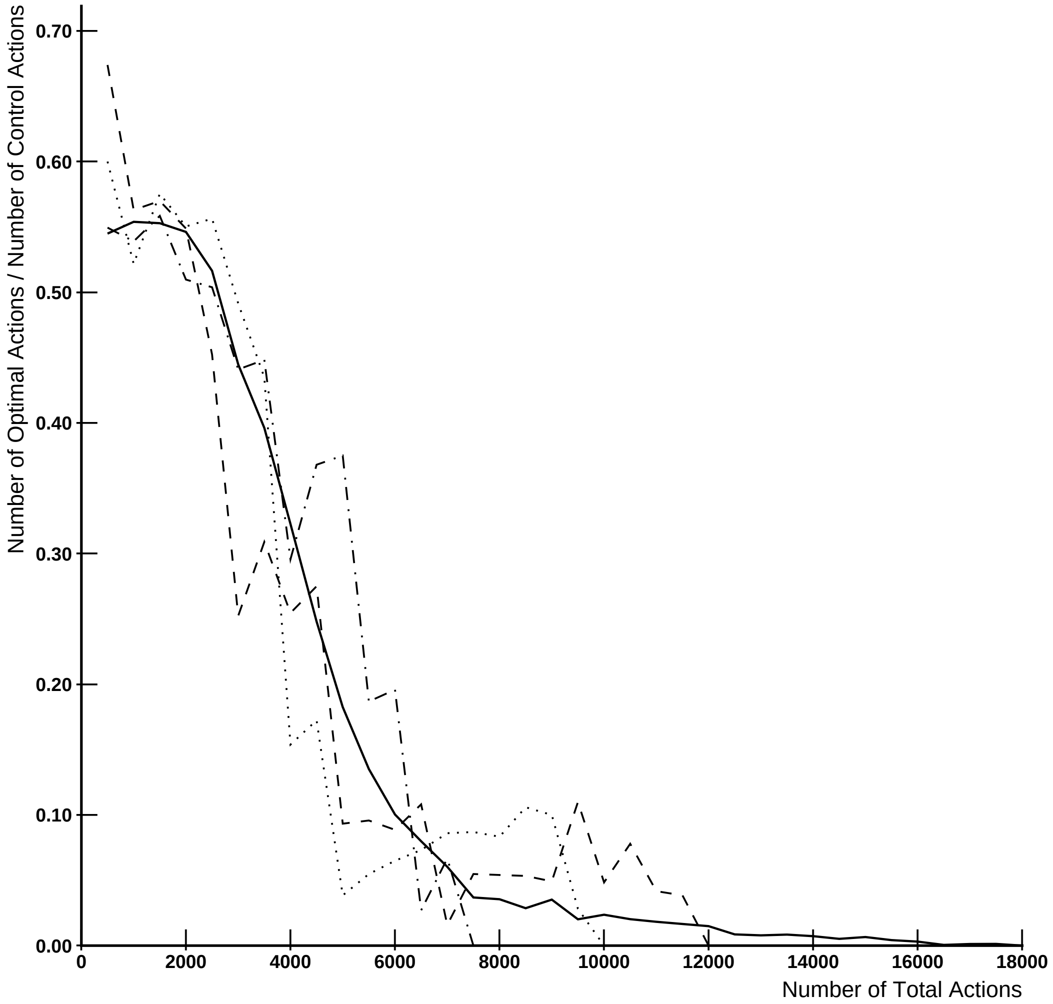

Figure 4.5. Learning Curves for the 4-Bit Shift Register Environment. The y-axis shows the fraction of suboptimal actions chosen during the past 500 time steps. The bold line shows the average performance over time for the control network over 25 replications. The dotted, dashed, and stippled lines show the performance over time for three different individual runs.

Jordan and Rumelhart address both of these issues. Their learning system is composed of a forward kinematics network and an inverse kinematics network. The inputs to the forward kinematics network are the joint angles $\boldsymbol{\theta}$, and the output is the position of the endpoint of the arm $\boldsymbol{x}$. The input to the inverse kinematics network is the position of the desired endpoint of the arm $\boldsymbol{x}$, and the outputs are the joint angles $\boldsymbol{\theta}$. Although the forward modeling technique will work with a variety of learning algorithms, we restrict our attention to the forward modeling method using back-propagation. In this case, the method involves differentiating the forward kinematics network in order to translate errors in cartesian space to errors in joint angles in order to provide training information for the inverse kinematics network. Their approach starts with a modeling phase, where the model is trained with random joint angle inputs and the resulting positions as targets. Once a reasonably accurate forward kinematics model has been learned, the output of the inverse kinematics model is connected to the input of the forward kinematics model, and the inverse kinematics model is trained by back-propagating through the forward kinematics model into the inverse kinematics model.

This technique, called *differentiating a model* by Barto [6], has some advantages over various other control acquisition approaches. Jordan and Rumelhart [35] note that this technique can identify a controller even if the controller is not well-defined (i.e., excess degrees of freedom). This approach has the potential to simultaneously differentiate the model and adjust the parameters of the model and controller. Furthermore, the derivative of the model's output with respect to its input can be efficiently computed by the back-propagation learning algorithm.

Our control acquisition architecture is related to an architecture called the *3-net* architecture that uses the idea of differentiating a model to obtain training information. The 3-net architecture shown in Figure 4.6 was proposed by Werbos [101]. The 3-net architecture consists of a controller, a model, and an adaptive critic. The controller maps states to actions, the model maps state/action pairs to next

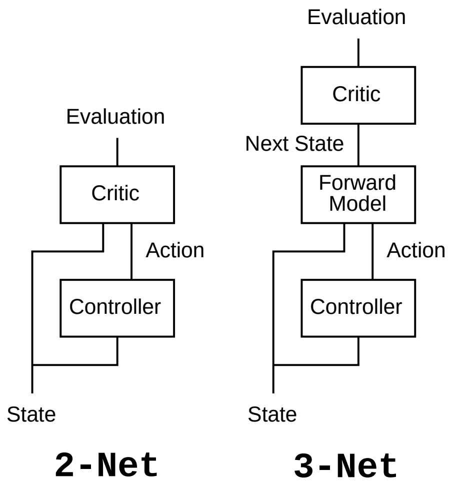

Figure 4.6. Control Architectures.

states, and the adaptive critic maps states to evaluations. The basic idea is to learn an evaluation function and then train the controller by differentiating this function with respect to the controller weights. These derivatives indicate how to change the controller's weights in order to minimize or maximize the evaluation function.

Our control acquisition architecture is a version of a 3-net architecture without an explicit controller. Instead of training a controller, our technique chooses actions based solely on the output of the adaptive critic; our technique does not involve an explicit representation of the control function. In the 3-net architecture the controller and adaptive critic are intimately related: the controller must adapt to changes in the adaptive critic and vice-versa. In contrast, in our architecture this relationship is simplified in that the adaptive critic directly defines the actions. In particular, the actions are chosen according to the current evaluation function—the action with the best evaluation is chosen.

It is useful to compare our architecture to the 2-net architecture shown in Figure 4.6 as proposed by Werbos [101] and Jordan and Jacobs [36]. The 2-net architecture consists of a controller and an action-dependent adaptive critic. The controller maps states to actions, and the action-dependent adaptive critic maps state/action pairs to evaluations. Jordan and Jacobs [36] show how to train a pole balancer using the method of differentiating the adaptive critic in a 2-net architecture. The adaptive critic learns to predict the reciprocal of time-until-failure given the current state and the controller's action. Given the current state as input, the controller produces a real-valued action that is interpreted as a force on the cart. The adaptive critic should output zero when the controller is performing as desired because in this case the time-until-failure should be infinite. Therefore, the desired output of the adaptive critic is zero, and the difference between the actual adaptive critic output and zero gives a distal error. This error can then be used to train the controller by back-propagating it through the weights of the adaptive critic, through the action output, and into the controller. In this way the controller can be changed so as to maximize the time until failure.

It has been said that it is easier to train a 2-net architecture than a 3-net architecture because there is no model (see Mahadevan and Connell [51]). One problem with building a model is that it is not sensitive to the particular control task. [4] Some features of the environment might be unrelated to the control task, and any technique that employs a model would have to spend a disproportionate amount of time learning about aspects of the environment that are irrelevant to the control task. With a 2-net architecture, an action-dependent adaptive critic is trained based on state/action input pairs, and a model is not required. But what if a model already exists, or if prior knowledge exists from which a model can be constructed off-line? Knowledge of relations between actions and their effects can be hardwired into the weights of a network environment model. In SLUG, the task of coding this knowledge is simplified because of the explicit representation of the effects of actions on a model's state as separate weight matrices. For example, symmetries in actions (such as action inverses) can be coded into a model. In the extreme case,

[4] On the other hand, building an environment model might be advantageous because a model could be used for many different tasks.

the entire model can be hardwired as illustrated in Chapter 4.4. Furthermore, if this is possible, it can significantly accelerate up the overall learning process because the adaptive critic learns an evaluation function based only on states and not state/action pairs as in the action-dependent adaptive critic.

Barto, Sutton, and Anderson [9] discuss a direct reinforcement method for control acquisition where the teacher provides only a reinforcement signal. The reinforcement signal indicates how well the controller is performing and not how to change the control signal. This is different than supervised learning, where the teacher provides a desired control signal. Barto, Sutton, and Anderson address the problem of learning to balance an inverted pendulum on a moving cart, where the reinforcement signal only indicates failure. This delayed reinforcement problem is particularly difficult because the consequences of the actions reveal themselves over time and interact with the consequences of other actions. This problem is solved by constructing an adaptive critic that learns to predict the discounted sum of all future penalties, where a penalty occurs only upon failure. The discounting gives greater weight to near-term penalties than to temporally distant penalties. The adaptive critic produces a reinforcement signal more informative than the reinforcement signal provided by the teacher. This new reinforcement signal gives the controller immediate feedback about the long-term consequences of its current action. In contrast to the technique of differentiating a model, the controller is trained using a simple reinforcement learning algorithm that uses the adaptive critic's output as the reinforcement signal.

Sutton [93] further develops the theory of learning algorithms appropriate for the training of adaptive critics. This class of algorithms, called temporal difference (henceforth referred to as TD) methods, address the problem of developing predictions in a delayed reinforcement setting. TD algorithms work well in multi-step prediction paradigms, where new information comes in at each step leading up to the outcome.

Sutton and Pinette [95] present a technique for learning a Markov model of stochastic maze environments with a connectionist network. This model is then used to learn the best path through a maze. The solution to the maze problem is learned much more quickly after initially learning a model of the maze. Their learning algorithm is based on the TD learning algorithm and produces a model that predicts the discounted expectation of the future sensations. Discounting gives greater weight to near-term sensations than to temporally distant sensations. The network is a fully connected recurrent network of linear units. These units encode the state, and unlike in the tasks considered in this thesis, the state is completely observable at each time step. The network is trained to construct a weight matrix that matches the environment's transition probability matrix. This network is used to make predictions by running the network forward until the predictions stabilize.

Barto, Sutton, and Watkins [10] (see also Watkins [97]) discuss the relationship between adaptive critic control techniques and stochastic dynamic programming and parameter estimation. The dynamic programming framework formalizes the adaptive critic technique (using TD methods) and suggests other learning methods.

4.4 Discussion of Other Possible Approaches

Our architecture provides the possibility of learning a model and control law simultaneously (although this technique is currently unimplemented in our system). This would allow the robot to learn in dynamic environments and would permit the use of a model for performing "mental experiments" as demonstrated by Sutton [94]. Sutton shows how a model can dramatically speed up the control acquisition process.

A variation of our control architecture can be used in environments where learning a model would be very time consuming. All components of the architecture remain the same, but the model would no longer be required to produce predictions of the sensations. Instead of predicting the next sensation values, the model would produce state information used by the adaptive critic to form an evaluation function. Consequently, the model would produce only state information necessary for the control task. In this case, the combination of an action-independent adaptive critic and model form an action-dependent adaptive critic. Judging from the superior performance of unconstrained SLUG over a "conventional" connectionist architecture, we would expect that unconstrained SLUG used in conjunction with an adaptive critic network would outperform one large, undifferentiated connectionist 2-net architecture (such as an Elman network).

In connectionist control acquisition, there are many viable alternative architectures and training procedures. However, at some level, all approaches require some sort of model of the environment in order to predict the consequences of actions. An explicit environmental model is especially important in environments whose state cannot be directly perceived. Thus a flexible, powerful model—such as the model constructed by SLUG—is an important component of a goal-seeking system operating in complex environments.