CONNECTIONIST MODELING AND CONTROL OF FINITE STATE ENVIRONMENTS

A Dissertation Presented

by

JONATHAN RICHARD BACHRACH

Submitted to the Graduate School of the

University of Massachusetts in partial fulfillment

of the requirements for the degree of

DOCTOR OF PHILOSOPHY

February 1992

Department of Computer and Information Sciences

CONNECTIONIST MODELING AND CONTROL OF FINITE STATE ENVIRONMENTS

A Dissertation Presented

by

JONATHAN RICHARD BACHRACH

Approved as to style and content by:

Andrew G. Barto, Chair

Michael C. Mozer, Member

Roderic A. Grupen, Member

Christopher V. Hollot, Member

W. Richards Adrion, Department Chair
Computer and Information Sciences

To such
supercalifragilisticexpeeallidoescious
parents,
Lou Baby and Santa Fe,
who made my life possible and
filled my life with love.
I love you.

Acknowledgements

I am indebted to Andy Barto for being such a superior advisor. Above all, Andy taught me how to write. He patiently read countless drafts and gave bunches of helpful comments.

I would like to thank the entire UMass Adaptive NetWorks group: Chuck Anderson, Steve Bradtke, Judy Franklin, Robbie Jacobs, Mike Jordan, Stephen Judd, Brian Pinette, Satinder, Vijay, and Richard Yee. They have helped me greatly. I would especially like to thank Robbie Jacobs and Brian Pinette. They both have been good friends and good critics. Brian has also helped me refine my ideas on robot navigation.

Dave Rumelhart has been very influencial and generous to me. I spent a year in his lab as a visiting graduate student. Most of the navigation ideas were developed with him while visiting his lab. I am constantly awed by his genious.

While in Dave's lab I benefited from my interactions with the other members of the PDP group: Bo Curry, Richard Durbin, Richard Golden, Chris Kortge, Ben Martin, Dan Rosen, Charlie Rosenberg, and Peter Todd. During that time I had the pleasure of working very closely with Richard Durbin. He was extremely helpful and I benefited enormously from my interactions with him. Richard is one smart man.

Mike Mozer and I have collaborated on ideas for several years now. He has been very instrumental in my thinking about the ideas presented in this dissertation. Furthermore, he has directly contributed some of these ideas, especially those presented in Chapter 2.4. Mike Mozer is one impressive thinking fellow.

Earlier in my graduate career I spent two summers working at GTE labs. There I had the honor of working with Rich Sutton. Above all, Rich taught me how to be a good scientist.

I had the pleasure of working with Mike Jordan while he was a post doc at UMass. His thoughts on motor control have been enormously influencial on me. Mike Jordan is one amazing renaissance fellow.

Finally, this research was supported by funding provided to Andrew G. Barto by the Air Force Office of Scientific Research, Bolling AFB, under Grant AFOSR-89-0526, and by the National Science Foundation under Grant ECS-8912623.

Abstract

CONNECTIONIST MODELING AND CONTROL OF FINITE STATE ENVIRONMENTS

February 1992

Jonathan Richard Bachrach,

B.S., University of California, San Diego

M.S., University of Massachusetts

Ph.D., University of Massachusetts

Directed by: Professor Andrew G. Barto

A robot wanders around an unfamiliar environment, performing actions and observing their perceptual consequences. The robot's task is to construct a model of its environment that will allow it to predict the outcome of its actions and to determine what action sequences take it to particular goal states. In any reasonably complex situation, a robot that aims to manipulate its environment toward some desired end requires an internal representation of the environment because the robot can directly perceive only a small fraction of the global environmental state at any time; some portion of the rest must be stored internally if the robot is to act effectively. Rivest and Schapire [72, 74, 85] have studied this problem and have designed a symbolic algorithm to strategically explore and infer the structure of finite-state environments. At the heart of this algorithm is a clever representation of the environment called an *update graph*. This dissertation presents a connectionist implementation of the

update graph using a highly specialized network architecture and a technique for using the connectionist update graph to guide the robot from an arbitrary starting state to a goal state. This technique requires a *critic* that associates the update graph's current state with the expected time to reach the goal state. At each time step, the robot selects the action that minimizes the output of the critic. The basic control acquisition technique is demonstrated on several environments, and it is generalized to handle a navigation task involving a more realistic environment characterized by a high-dimensional continuous state-space with real-valued actions and sensations in which a simulated cylindrical robot with a sensor belt operates in a planar environment. The task is short-range homing in the presence of obstacles. Unlike many approaches to robot navigation, our approach assumes no prior map of the environment. Instead, the robot has to use its limited sensory information to construct a model of its environment. A connectionist architecture is presented for building such a model. It incorporates a large amount of a priori knowledge in the form of hard-wired networks, architectural constraints, and initial weights. This navigation example demonstrates the use of a large modular architecture on a difficult task.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

A robot is placed in an unfamiliar environment. It explores the environment by performing actions and observing their perceptual consequences. The robot's task is to construct an internal model of the environment that will allow it to predict the outcome of its actions and to determine what sequences of actions it should take to reach particular goal states. This scenario is extremely general, applying not only to physical environments, but also to abstract and artificial environments such as electronic devices (e.g., a VCR, phone answering machine, car radio), computer programs (e.g., a text editor), and classical AI problem-solving domains (e.g., the blocks world). Any agent—human or computer—that aims to flexibly manipulate its environment toward some desired end requires an internal representation of its environment because, in any reasonably-complex situation, the agent can directly perceive only a small fraction of the global environmental state at any time; some portion of the rest must be stored internally if the agent is to act effectively.

The goal of this thesis is to develop connectionist learning algorithms that can efficiently construct models of, and controllers for, unfamiliar environments. In each environment, the robot has a set of *actions* it can execute to move from one environmental state to another. At each environmental state, a set of *sensations* can be detected by the robot. The sensation values comprise the robot's *view*.

A *model* is used to predict the environment's behavior, and a *controller* is used to determine sequences of actions that efficiently lead to desirable environmental states.

The modeling task involves constructing a model that mimics the behavior of the environment in the sense that, given an identical sequence of actions, the model will yield predictions of the sensations that match those produced by the environment. Although a model cannot be constructed without carefully exploring the environment, the emphasis of this thesis is not on the exploration problem, and the most primitive exploration strategies are assumed. The control task involves producing a minimal length sequence of actions to reach a particular goal state.

1.1 Partial State Information

We assume that an environment is a discrete-time dynamical system with finite state set, and we assume the robot has only partial knowledge of the state set, the state transition function, and the output function. Furthermore, we assume the robot has only limited knowledge of the current environmental state at any time step. One central focus of this thesis is on environments with incompletely observed states. Any particular view can be generated from more than one environmental state; that is, the mapping from environmental states to views is many-to-one. Whitehead and Ballard [102] call this problem *perceptual aliasing* and call a view *perceptually ambiguous* if it can represent multiple environmental states.

Model building and control are particularly difficult when the robot does not have access to the complete state of the environment. In many cases, the robot's current view will not suffice for the particular task because the model and controller might have to output different values in different environmental states having the same or similar views. As an example, consider the problem of learning to predict the view after taking a particular action in the 4×4 checkerboard environment shown in Figure 1.1. Half the squares in the environment yield the same sensations and are thus perceptually ambiguous (assuming the robot sees only one square at a time). For example, the two squares labeled A and B in Figure 1.1 yield exactly the same view, and without some mechanism to disambiguate these squares, it is impossible

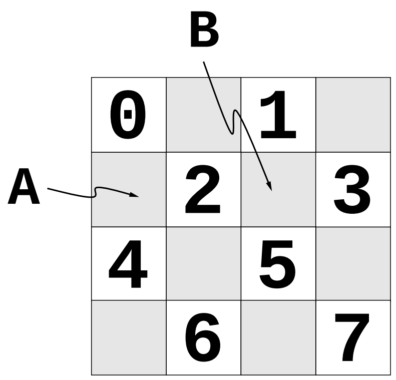

Figure 1.1. An Example of the Need for Internal State in the $n \times n$ Checkerboard Environment. The checkerboard environment consists of an $n \times n$ grid of squares. The robot occupies a particular square and can sense the landmark in that square, if any. There are a number of unique landmarks distributed in a checkerboard pattern across the grid. These are depicted as the numbers in the squares. Half of the squares have no landmark and are indistinguishable from each other. At each time step the robot can move either up, down, left, or right one square. Movement off one edge of the grid wraps around to the other side.

to learn a model of this environment. Without such a mechanism, a model must predict—incorrectly—that the next view after moving right starting in the squares labeled A and B will be the same; whereas in fact, the actual next view would be landmark 2 and landmark 3 respectively. In order to solve this problem, the learning system must represent the state of the environment in some fashion.

1.2 Modeling

A learning system can represent the environment's state by constructing a model of the environment. The learning system is trained to produce a model with *internal states* that correspond to environmental states and that serve to distinguish the perceptually ambiguous environmental states. A *state representation* captures the salient aspects of the unbounded history of the environment's behavior necessary to predict the environment's future behavior. We present two canonical representations that correspond to the extremes of the representational spectrum. These are the *delay-coordinate* representation and the *state-space* representation. (These two techniques are discussed in greater detail in Chapter 1.5.) Each succinctly encodes an environment's past behavior.

In the delay-coordinate representation, the environment's state is encoded as a fixed number of past actions and views. Engineering techniques that use this representation include adaptive filters [103] and DARMA (Deterministic Auto Regressive Moving Average) models [30]. Note that some FSA's cannot be accurately modeled using a delay-coordinate state representation. (See Chapter 1.5 for an example of such an FSA.)

Because of this limitation, we restrict our attention to the state-space representation, where the state of the environment is represented in the model as the value of a vector of state variables. In this case, the state representation corresponds to the particular encoding of the environmental state by the state variables. The many state-space approaches for building models of finite state environments can be compared and contrasted along two main dimensions that refer to the state representation. The first dimension pertains to the range of values that can be taken by each state variable. Two representations are distinguished: the *discrete* representation in which the range of each state variable is a finite set, and the *continuous* representation in which the state variales are real-valued. For the purposes of this discussion and without loss of generality, the state variables in the discrete representation are Boolean variables. The second dimension concerns how states are assigned to patterns of values of the state variables. Again two representations are distinguished: the *localist* representation, which assigns each environmental state to an individual state variable, and the *distributed* representation, which encodes each environmental state as a *pattern* of values across multiple state variables. Table 1.1 summarizes the four possible combinations of range and encoding distinctions we have made. In the following text we are concerned with three of these, which

Table 1.1. A Two-Dimensional Categorization of the Various State Representations for Finite State Environments. The first dimension refers to the range of each state variable, and the second dimension refers to the encoding of state across state variables. Machines that update the values on these state variables are listed in three of the entries.

	Localist	Distributed
Discrete	State Transition Graph Machine	Update Graph
Continuous		State-Space Model SLUG

we call the discrete-localist approach, the discrete-distributed approach, and the continuous-distributed approach.

It is equally important to discuss the mechanism by which the values of state variables are calculated. For the purposes of this dissertation, we consider a restricted class of machines that update the values of the state variables. (For now we do not consider the important problem of how the robot's sensations can be predicted based on the internal state of the machine.) The state variables' values are updated using a network of primitive machines, where each machine updates one of the state variables. Each primitive machine computes the value of its state variable as a function of the robot's chosen action and the values of state variables associated with other machines in the network to which it is connected.

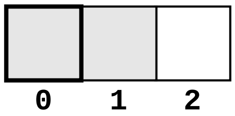

Figure 1.2. The 3-Bit Shift Register Environment. White squares indicate that the bit is on, and shaded squares indicate that the bit is off. The leftmost square is enclosed in a bold frame to symbolize that it is a sensation.

We describe an example machine for each main representational approach shown in Table 1.1 We call these machines the discrete-localist machine, the discrete-distributed machine, and the continuous-distributed machine. Note that although continuous-distributed machines are not FSA's, they can represent the behavior of FSA's. In order to facilitate comparisons, we define the size of a machine to be the number of primitive machines of which it is composed plus the number of connections between the primitive machines. We illustrate these various machines and their associated representations by considering the 3-bit shift register environment shown in Figure 1.2. In this environment, the robot senses the value on the leftmost bit of an n-bit shift register, can rotate the register left or right, and can flip the leftmost bit.

In a discrete-localist machine for the 3-bit shift register, each state of the register is represented by a state variable. The value of the state variable corresponding to state q of the register is one if the register's state is q and is zero otherwise. Thus, only one state variable has value one at any time step. The values of state variables in this discrete-localist representation can be updated using a *state transition graph machine*, where each node in the graph is a primitive machine with one state variable. Labeled links connecting each node of the graph to other nodes correspond to the actions that the robot can execute to move between environmental states.

Associated with each node is a set of discrete-valued sensations that can be detected by the robot when the environment is in the corresponding environmental state. Figure 1.3 shows the state transition graph machine for the 3-bit shift register

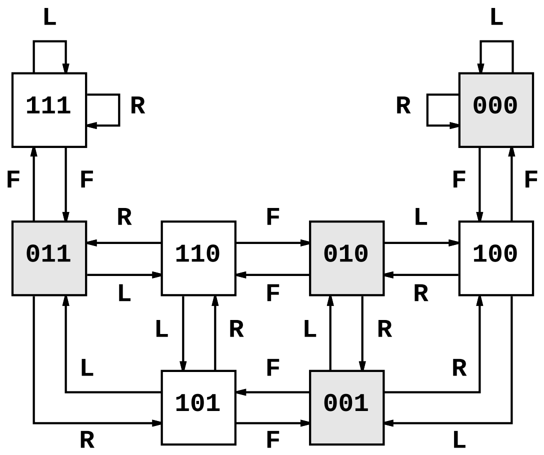

Figure 1.3. The State Transition Graph Machine for the 3-Bit Shift Register Environment. Each box represents a state with the shade of the box coding the single sensation value for that state. A shaded square indicates that the leftmost bit is off, a light square that the leftmost bit is on. The transitions are depicted as arrows labeled with the actions, where `L` is shift left, `R` is shift right, and `F` is flip the leftmost bit.

environment. Each node is identified by the corresponding environmental state, labelled by the settings of bits in the shift register. This environment has $2^3 = 8$ states, one for each of the possible settings of the bits in the register. Here, there is only one sensation, the value of the leftmost bit, which is depicted in Figure 1.3 by the shade of the node. Each link between nodes is labeled with one of the three actions: flip (`F`), shift left (`L`), or shift right (`R`).

The state transition graph machine is one way to represent the behavior of the environment. If the state transition graph machine is known, one can predict the sensory consequences of any sequence of actions. Further, the state transition graph machine can be used to determine a sequence of actions needed to obtain a certain goal state. For example, if the robot wishes to turn on all the bits in the register, it should follow a link or sequence of links in the state transition graph machine that lead to the node in which all the bits are on.

Although one might try developing an algorithm to learn a state transition graph machine, Schapire (in ref. [85]) presents several arguments against it. [1] Most importantly, the state transition graph machine does not take advantage of all the regularities present in the environment. For example, in the n-bit shift register, the action `F` has the same behavior independently of the value of the unobservable bits (i.e., the bits other than the leftmost bit), yet in the state transition graph machine of Figure 1.3, knowledge about `F` must be separately encoded for each bit and in the context of the particular values of the other bits. Thus, the simple semantics of an action like `F` must be encoded repeatedly for each of the 2^n distinct states. The size of a state transition graph machine is insensitive to this kind of environmental regularity, and consequently, the running time of any learning algorithm that produces a localist machine is equally insensitive.

As an alternative, one could construct a *distributed* machine that utilizes fewer primitive machines than environmental states, where the representation of each environmental state is distributed across the state variables of these primitive machines. The hope is that the distributed machine would be smaller than the size of the state transition graph machine, that is, there would be fewer state variables and fewer connections needed. In the shift register environment, the bits of the shift register form a discrete-distributed state representation. A distributed machine using this state representation would have n state variables instead of the 2^n needed for the state transition graph machine. (Distributed machines are discussed in more detail below, in Chapter 2.4, and in the Appendix.)

One common approach (cf. Kohavi [44]) is to first construct the state transition graph machine, and using this graph, to then create a more compact distributed machine. Although the end product of this process is a smaller machine, the problem

[1] Schapire actually presents arguments against developing an algorithm to learn a state transition graph, but the same arguments apply to algorithms for learning state transition graph machines.

is that the learning time would still be a function of the size of the state transition graph machine instead of the size of the more compact distributed machine. [2]

A better approach—the approach taken in this thesis—is to learn a distributed machine directly. Rivest and Schapire [72, 74, 85] developed a discrete-distributed machine, called an *update graph*, that is a network of primitive FSA's. The update graph maintains a particular discrete-distributed state representation, called the *diversity representation*. (The details of the diversity representation and the update graph are presented in Chapter 2.4 and in the Appendix.) Rivest and Schapire developed a learning algorithm that explores the environment and incrementally constructs an update graph.

The third approach to constructing an environment model is the continuous-distributed approach. This approach is typified by traditional engineering methods, where the problem of constructing a model is called *system identification*. We consider one particular continuous-distributed machine from the engineering literature called a *state-space model*. Other engineering approaches are discussed in Chapter 1.5. In contrast to the primitive FSA machines used in the discrete approach, the primitive machines used in the traditional state-space model calculate the values of their state variables as real-valued linear functions of the selected action and the values of the state-variables of the machines to which they are connected. These functions are constructed using a set of real-valued parameters, which we called *weights*. The number and interconnectivity of these primitive machines is fixed a priori. Constructing a state-space model amounts to first choosing the number and interconnectivity of the primitive machines and then choosing a set of weights that gives the best prediction performance. This last step is called *parameter estimation* in the engineering literature. It differs significantly from Rivest and Schapire's symbolic learning algorithm

[2]The usual approach actually produces the state transition graph as the intermediate result, but the complexity arguments still apply.

which constructs a graph of primitive machines by incrementally adding new machines and their connections.

Our connectionist method, called *SLUG*, performs Subsymbolic Learning of Update Graphs. It is related to both the Rivest and Schapire approach and the traditional engineering approaches. We show that our connectionist method offers complementary strengths and new insights into the problem of building a model of a finite state environment. SLUG is a continuous-distributed method utilizing a connectionist network that shares many of the qualities of the engineering approach: it uses real-valued state variables, a fixed network topology, parameterized primitive machines, and parameter estimation to select weight values. In contrast to the traditional engineering approach, SLUG uses primitive machines, called *units*, that calculate the values on their state variables using a nonlinear function. We discuss the general connectionist approach in more detail in the Section 1.2.2, and then discuss SLUG in Section 1.2.3.

1.2.1 State Estimation and Real-World Issues

Before going into more detail, we first address a problem related to the construction of an environment model: the problem of setting a model's state to most closely correspond to the environmental current state. This problem is called *state estimation* in the engineering literature, and it is necessary because only part of the environmental state or a function of the environmental state is directly measurable and/or the available state information is noisy. The engineering approach views this as an optimization problem. Rivest and Schapire propose a different approach for solving this problem in the discrete case, and we discuss their technique in the Appendix.

It is important to point out that parameter and state estimation are particularly difficult for autonomous robots acting in the real world (cf. Kaelbling [38]), where the robot is continually receiving inputs and executing actions in its environment.

The robot must be able to keep up with the important events in the environment and thus cannot spend an unbounded amount of time in computing each action. Therefore, any algorithm must operate within a fixed time frame. In addition, an autonomous robot generally neither has access to a *reset* command, which places the environment in a known state, nor an *undo* command, which restores the state of the environment to the state existing prior to the execution of the last action. Thus, a state estimation algorithm must perform while the robot is continually interacting with its environment.

1.2.2 Connectionist Networks

The proposed connectionist approach differs from the traditional engineering approach by using a novel class of parameterized nonlinear network models. Connectionist networks are composed of simple differentiable nonlinear processing elements called *units*. The units are connected to each other and pass activity through weighted connections. Each unit computes its activity as a nonlinear function of a set of parameters, called *weights*, and its inputs; its output is transmitted to other units to which it is connected. The units, connections, and weights constitute a connectionist *network*.

Gaussian and logistic units are two popular unit types. A Gaussian unit computes its output as follows:

$$y = e^{\frac{-\sum_j (w_j - x_j)^2}{\sigma^2}}, \tag{1.1}$$

where y is the output of the unit, w_j is the j^{th} weight of the unit, x_j is the j^{th} input, and σ is the variance parameter. The logistic unit computes its output as follows:

$$y = \frac{1}{1 + e^{-\sum_j x_j w_j}}. \tag{1.2}$$

The activities of the *input units* serve as the inputs to the network, while the activities of the *output units* provide the output of the network. Units that are neither input nor output units are called *hidden units*. *Feedforward* networks are networks without cycles in their connection graphs. *Recurrent* networks have cycles, that is, they contain feedback connections in addition to feedforward connections.

Connectionist networks have a number of attractive properties (as pointed out in ref. [6]) that make them appropriate for learning models of, and controllers for, nonlinear systems. They can be implemented in fast parallel hardware, gradient calculations useful for parameter estimation can be computed very efficiently with the back-propagation algorithm (developed by le Cun [48], Parker [65], Rumelhart, Hinton, and Williams [79], and Werbos [99]), and learning rules for connectionist networks are predominantly on-line algorithms. All these properties make networks attractive for real-time processing. Given enough units, networks can represent any function to any desired degree of accuracy (Hornik, Stinchcombe, and White [32]). Because of the statistical nature of connectionist learning rules, connectionist networks are able to learn in spite of noise in the examples, and they can also learn to approximate probabilistic functions in a principled fashion [31]. Furthermore, connectionist networks provide a variety of methods for encoding prior knowledge in the form of particular architectures, weight constraints, and initial weights. Finally, connectionist networks provide the flexibility to enhance the representation of the inputs without modifying the basic learning procedure. For example, the basic back-propagation learning algorithm applies to a large class of networks with various network architectures and unit types.

1.2.3 Our Modeling Approach

Our basic learning architecture is a recurrent network as shown in Figure 1.4. The network is composed of a fixed number of units, called *state units*, that serve as the state variables of the model. Some of these units are called *observable state units* and serve to predict the sensations. The remaining state units are called *hidden state units* and serve to encode state information useful for predicting the sensations and disambiguating perceptually ambiguous environmental states. The state units

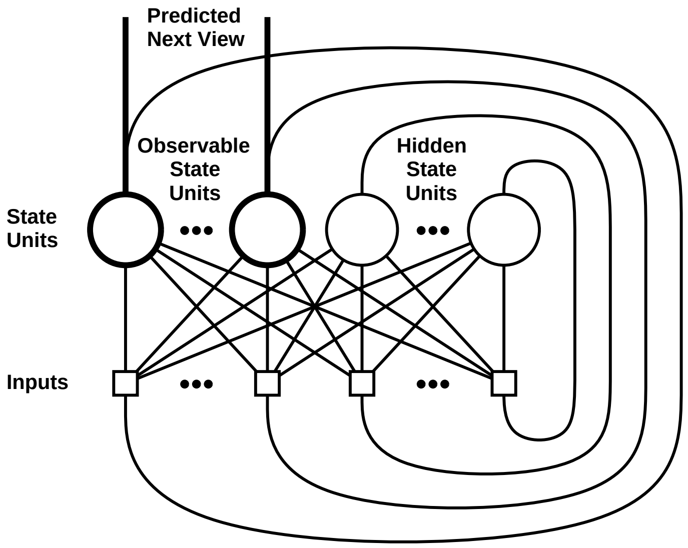

Figure 1.4. The SLUG Architecture. In this architecture there is one set of weights for each action. The weights are determined by the current action.

are fully connected to each other yielding a fully-connected recurrent network. Each unit has a separate set of weights for each of the robot's available actions, and the weights at any moment are determined by the chosen action. The outputs of the network are computed using a weighted sum involving the current weights (i.e., the weights for the chosen action) and the previous output of the state units. Although nonlinearities are introduced by the operation of choosing a set of weights based on the chosen action, the behavior of the network at any one time step is linear.

The technique called *back-propagation in time* [79] is used to modify the weights of the network to improve prediction performance. It is an algorithm for calculating the gradient of an error measure with respect to the weights of recurrent networks. In this case, the error measure is the squared error between the predicted and the actual sensation values.

In order to produce better state estimates, the actual instead of the predicted values of the sensations are used for computing the model's next state, where the sensations are assumed to be noise-free. In keeping with Williams and Zipser [104], this technique is called *teacher forcing*. The combination of both back-propagation in time and teacher forcing provides a means for performing parameter and state estimation simultaneously.

A unique feature of this type of network architecture is that a special case of it, called SLUG, can learn an update graph realization of an FSA. An update graph has a natural and direct connectionist implementation. It has a number of beneficial properties that differentiate it from typical connectionist representations of FSA. After training, because the activities and weights are discrete, SLUG can perfectly predict the sensations infinitely far into the future. Additionally, the state units are readily interpretable within the update graph formalism. Many formal and empirical results exist (cf. Rivest and Schapire [73, 85]) that lend intuition and rigor.

Others have taken similar approaches to the modeling problem. Our approach is related to more traditional engineering approaches such as state-space and DARMA models. These relations are elaborated in Chapter 1.5. The idea of back-propagation in time has been used to model nonlinear dynamical systems [79]. Recently, a real-time variant of this procedure was developed and demonstrated by Bachrach [3], Mozer [55], Robinson and Fallside [76], and Williams and Zipser [104]. Rivest and Schapire [73, 85] explore a learning algorithm for inferring update graphs using a discrete-distributed representation (as opposed to the continuous-distributed representation used in this thesis). Giles et al. [26] discuss a bilinear recurrent network for learning regular languages. Whitehead and Ballard [102] discuss an alternative approach for dealing with the problem of perceptually ambiguous environmental states. In their formulation, they separate the perceptual state from the environmental state. For instance, the environmental state might be the position of the robot in its environment, and the perceptual state might be the orientation of the robot's sensors. An unrealistic requirement on their approach—one that does not hold in the environments that we consider—is that each environmental state must have at least one perceptual state that yields an unambiguous view, that is, a view unique to that environmental state. Unlike their approach, our technique does not require this assumption and will work in environments where their technique fails. Chapter 2.4 expands on the relationship between the proposed approach and similar approaches taken by others. Mozer and Bachrach have reported on some of the work presented in this dissertation [57, 58, 60, 59, 5].

1.3 Control Acquisition

Having discussed methods for learning a model of the environment, we turn to a control acquisition architecture that can use this model to determine a minimal length sequence of actions to follow to reach a particular goal state. We call the particular control task addressed in this thesis the *homing task* (which is an example of what is called an *optimal regulation task* in the control literature). It involves producing actions that cause the environment state to change from an arbitrary starting state to a given goal state in minimum time. A technique is described for training a controller to perform the homing task. This training process is referred to as *control acquisition.*

The robot acquires the desired homing behavior using a learning paradigm called *reinforcement learning.* It is useful to distinguish this paradigm from another learning paradigm called *supervised learning.* In supervised learning, the robot is given desired actions, whereas in reinforcement learning, the robot is given only reinforcement signals, which do not directly specify the desired actions. In the latter case, the robot must estimate the desired action by comparing given reinforcement signals for various actions taken over time.

As an additional difficulty, the robot must solve a sequential decision task whose objective is to maximize a performance measure that evaluates its behavior over an extended period of time. Although the performance measure evaluates the efficiency of a homing trajectory, the robot is given only a signal that indicates when it is located at the goal. From this limited reinforcement signal, the robot must determine a minimal length sequence of actions necessary for homing. This poses a difficult

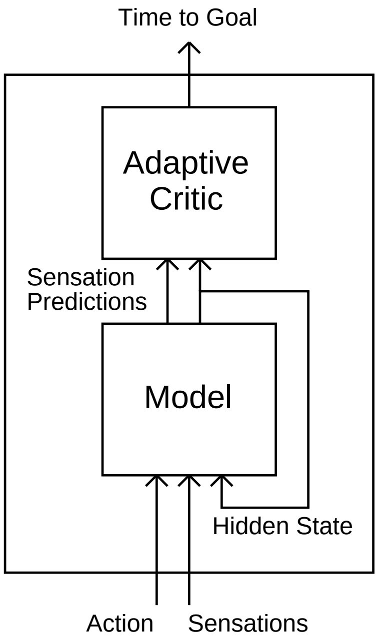

Figure 1.5. Connectionist Control Acquisition Architecture.

credit assignment problem because the consequences of the actions reveal themselves over time and interact with the consequences of other actions.

1.3.1 Partial State Information

In order to make the control acquisition task more realistic, we consider the impact of partial state information on control acquisition. As an example of the difficulty of partial state information, consider the problem of learning to go to the home square, labeled 2, in the 4×4 checkerboard environment shown in Figure 1.1. As before, the two squares A and B on opposite sides of square 2 yield exactly the same view. Without some mechanism to disambiguate these squares it is impossible to learn an optimal control rule for this environment.

To handle the problem of partial state information we propose a control acquisition architecture that uses the model described in Section 1.2.3 to determine a minimal sequence of actions required to reach a goal state. The architecture consists of a *model* and an *adaptive critic* as shown in Figure 1.5. The adaptive critic associates the model's current state with the expected time to reach the goal state. At each time step, the action is chosen which minimizes the output of the adaptive critic. The adaptive critic is trained using a *temporal difference (TD)* method to make successive predictions of time to reach the goal state decrease by one and to make the prediction at the goal state zero.

The proposed control acquisition technique was motivated by the work of other researchers. Barto, Sutton, and Anderson [8] and Sutton [92, 93] developed the temporal difference method which is related to a technique invented by Samuel [83]. Sutton, Barto, and Anderson [8] and Barto [6] demonstrate a direct reinforcement learning technique involving the temporal difference method for training a controller. Watkins [97], Sutton [92], Barto [6], and Kaelbling [38] discuss reinforcement learning issues and methods. Whitehead and Ballard [102] discuss the partial state information problem in control acquisition and propose a mechanism to handle it. Werbos [100] developed a control network, called the *3-net architecture*, involving an adaptive critic, model, and controller which is most similar to the proposed control network. Jordan and Jacobs [36] and Werbos [100] discuss a related control network, called a *2-net architecture*, involving an adaptive critic and a controller. These and other related approaches are elaborated in Chapter 3.7.

1.4 Navigation

In chapter4.4, we describe simulations of a cylindrical robot with a sonar belt in a planar environment. The navigation task is short-range homing in the presence of obstacles. A control acquisition architecture is used for navigation which incorporates a large amount of a priori knowledge in the form of hard-wired networks, architectural constraints, and initial weights. This architecture, with critic and model, is basically the same as shown in Figure 1.5. This example demonstrates the use of a large modular architecture on a difficult task.

We study this task in order to consider the complexities of more realistic environments characterized by high-dimensional continuous state-spaces and real-valued actions and sensations. This portion of this dissertation does not address the problem of partial state information: we assume that all environmental states are perceptually unambiguous. We make this simplification in order to bring to the foreground the issues of control acquisition in high-dimensional continuous state-spaces. Of course, we would eventually like to merge the techniques developed in this portion of this dissertation with those developed to handle partial state information.

The navigation task requires the solution of a number of problems. The sensory information forms a very high-dimensional continuous space, and successful homing generally requires a nonlinear mapping from this space to the space of real-valued actions. Furthermore, training networks with logistic hidden units is not easy in this problem. Instead we use networks with Gaussian hidden units, but because the state space is high-dimensional and continuous it is impractical to just uniformly distribute the receptive fields of these Gaussian units throughout the state space. Instead we use Gaussian units whose initial weights (and thus receptive fields) are determined using expectation maximization. This is a soft form of competitive learning (cf. Nowlan [63] and Durbin [19]) that, in our case, creates spatially-tuned units.

Unlike many approaches to robot navigation, our approach assumes no prior map of the environment. The robot has to use its limited sensory information to construct a model of its environment. Given this sensory information, models of the robot's environment are much more difficult to learn than models for the FSA environments discussed in Section 1.1. For this reason we use a hard-wired model whose performance is good in a wide range of environments. Here the philosophy is to learn only things that are difficult to hard-wire.[3] Finally, it is difficult to reach home using random exploration, thereby making simple trial-and-error learning intractable.

[3]Note that, in the future, when we consider even more realistic environments involving perceptual ambiguity and/or realistic sensors, it will be necessary to construct models.

The solution to this problem involves building a nominal initial controller that chooses straight-line trajectories home.

Our approach is related to other navigation research as follows. The basic approach is strongly related to our control acquisition approach and related research (see Section 1.3). Our approach is also related to a more traditional technique for navigation that uses potential functions (see Latombe [47] for an overview). Briefly, our approach differs from the usual potential function technique in two fundamental ways. First, our technique involves learning and can adapt the robot's behavior to produce efficient trajectories. Second, our technique does not require explicit object models; it constructs representations directly from the sensory information. Our approach is similar to the subsumption architecture proposed by Brooks [13]. Our approach and that of Brooks both involve the orchestration of multiple behaviors through the use of multiple experts, but in contrast to the subsumption architecture, ours involves learning and continuous mappings constructed with connectionist networks. Recent work by Mahadevan and Connell [51] and Maes and Brooks [50] propose preliminary learning techniques for the subsumption architecture. See Chapter 4.4 for an expanded discussion of related research.

1.5 Organization of the Dissertation

This dissertation elaborates methods and presents empirical results concerning modeling and control. Chapter 1.5 describes related research by others addressing the issues of modeling FSA environments with partial state information. Chapter 2.4 presents our approach to modeling FSA environments with partial state information. Chapter 3.7 presents a method of control acquisition for these environments. Chapter 4.4 describes an approach to learning to navigate in a simulated environment. Chapter 5.8 summarizes the research presented in this dissertation, briefly discussing the contributions, problems, and directions for future research. The Appendix discusses the update graph in more detail.