Reinforcement Learning and Its Application to Control

Vijaykumar Gullapalli

COINS Technical Report 92-10

January 1992

REINFORCEMENT LEARNING AND ITS APPLICATION TO CONTROL

A Dissertation Presented

by

VIJAYKUMAR GULLAPALLI

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

REINFORCEMENT LEARNING AND ITS APPLICATION TO CONTROL

A Dissertation Presented

by

VIJAYKUMAR GULLAPALLI

Approved as to style and content by:

[illegible]

Andrew G. Barto, Chair

[illegible]

Roderic A. Grupen, Member

[illegible]

B. Erik Ydstie, Member

[illegible]

Michael I. Jordan, Member

[illegible]

W. Richards Adrion, Department Chair
Computer and Information Sciences

To

my parents,
Mani Devi and Subrahmanyam Venkata,

my sisters,
Sharada and Lata,

and my brother,
Ravi Kumar.

ACKNOWLEDGEMENTS

Andrew Barto has been a constant source of inspiration to me throughout my tenure as a graduate student. His excellent technical expertise in an incredible range of disciplines makes him unique among researchers and unique as an advisor. I have benefited tremendously from Andy's guidance, enthusiasm, constructive criticism, and support—technical as well as financial. With great patience, Andy also taught me to communicate my ideas concisely and coherently in writing. His influence on both my thinking and my writing is evidenced throughout this dissertation. Michael Jordan hired me as a research assistant several years ago when I was a naive beginning graduate student. Since then, I have gained much from Mike's academic virtuosity and his ability to define important research issues in a clear fashion. Mike taught me the importance of maintaining a broad range of research interests and showed me how to precisely formulate research questions. Mike has also constantly challenged my ideas and my thinking and has forced me to back my intuition with sound arguments and hard evidence. I am also grateful to Mike for making available his network simulation software, which I used for several of the simulations reported in this dissertation.

I am also grateful for the help I received from Roderic Grupen and Erik Ydstie. I thank Rod Grupen for his help and support in my forays into robotics. His encouragement and guidance were mainly responsible for my daring to move beyond simulations and address "real-world" control problems. Rod also provided the LaTeX macros used to format this thesis, thereby saving me countless hours of painful formatting. Erik taught a very useful course in adaptive control and provided useful information about Recursive Least Squares. Erik also clarified some of the relationships between connectionist learning control and traditional control.

I am also grateful for many useful interactions, academic and otherwise, that I have had with my colleagues at UMass. In particular, I would like to thank Jonathan Bachrach, Neil Berthier, Steve Bradtke, Robbie Jacobs, Steven Judd, Brian Pinette, Sharad Saxena, Rahul Simha, Satinder Singh, and Richard Yee. My research has also benefited from interactions with Chuck Anderson, Judy Franklin, Rich Sutton, and others at GTE Labs. Kamal Souccar's considerable expertise in the nuts and bolts of robotics stood me in good stead when I was experimenting with the Zebra Zero.

My parents, brother, and sisters somehow endured my seemingly endless tenure as a graduate student and, what's more, provided me encouragement and support every step of the way. Thanks guys!

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

REINFORCEMENT LEARNING AND ITS APPLICATION TO CONTROL

February 1992

Vijaykumar Gullapalli,

B.S., Birla Institute of Technology and Science, India

M.S., University of Massachusetts

Ph.D., University of Massachusetts

Directed by: Professor Andrew G. Barto

Learning control involves modifying a controller's behavior to improve its performance as measured by some predefined *index of performance* (IP). If control actions that improve performance as measured by the IP are known, supervised learning methods, or methods for learning from examples, can be used to train the controller. But when such control actions are not known a priori, appropriate control behavior has to be inferred from observations of the IP. One can distinguish between two classes of methods for training controllers under such circumstances. Indirect methods involve constructing a model of the problem's IP and using the model to obtain training information for the controller. On the other hand, direct, or model-free, methods obtain the requisite training information by observing the effects of perturbing the controlled process on the IP. Despite its reputation for inefficiency, we argue that for certain types of problems the latter approach, of which reinforcement learning is an example, can yield faster, more reliable learning. Using several control problems as examples, we illustrate how the complexity of model construction can often exceed that of solving the original control problem using direct reinforcement learning methods, making indirect methods relatively inefficient. These results indicate the importance of considering direct reinforcement learning methods as tools for learning to solve control problems. We also present several techniques for augmenting the power of reinforcement learning methods. These include (1) the use of local models to guide assigning credit to the components of a reinforcement learning system, (2) implementing a procedure from experimental psychology called "shaping" to improve the efficiency of learning, thereby making more complex problems amenable to solution, and (3) implementing a multi-level learning architecture designed for exploiting task decomposability by using previously-learned behaviors as primitives for learning more complex tasks.

TABLE OF CONTENTS

List of Tables

List of Figures

CHAPTER 1
INTRODUCTION

Although problems in controlling complex processes are usually considered the exclusive purview of control theory, many control problems of current importance suggest methods combining control theory and artificial intelligence (AI). The increasing complexity of these control problems, arising from nonlinear, stochastic, or non-stationary process behavior, makes them less amenable to the rigorous analysis customary in traditional approaches to control systems design. Under such circumstances, heuristic approaches to solving these control problems are finding greater currency. At the same time, many heuristic problem-solving techniques are being developed in AI for "situated" intelligent systems, i.e., systems that operate in a tightly coupled fashion with their environments. Game-playing systems are perhaps the most well-studied examples of such systems. Because situated systems must meet performance objectives similar to those specified for controllers, problem-solving techniques developed for situated systems can also be useful for solving control problems. In addition, applying techniques from AI to control problems can illuminate the strengths and weaknesses of these techniques and suggest ways of improving them.

In this dissertation, we address some of the issues that arise in *learning* to control complex processes. Learning control involves modifying the controller's behavior to improve its performance as measured by some predefined *index of performance* (IP). If control actions that improve performance as measured by the IP are known, supervised learning methods, or methods for learning from examples, can be used to train the controller. But when such control actions are not known a priori, appropriate control behavior has to be inferred from observations of the IP. Two different classes of methods can be used for deciding how to modify a controller's behavior based on the IP. *Indirect*[1] methods involve constructing a model of the IP in a form that can be used to supply training information to the controller. The application of indirect methods to learning control problems therefore involves two distinct operations: construction of an adequate model, which can itself be regarded as a learning problem, and using the model to train the controller. As alternatives to this, *direct*, or model-free, methods rely on perturbing the process and observing the consequences on the IP to obtain the required training information. Process perturbations can be caused by active perturbation of the control actions or by disturbances (of possibly unknown characteristics) from a source external to the controller.

Indirect methods are widely used in AI, often with hand-crafted "knowledge bases" being used instead of learned models. The only prominent examples of direct learning methods, however, are those developed for learning tasks in which the learning system has to infer appropriate actions based solely on evaluative feedback. Because of the similarity of such learning tasks to trial-and-error learning studied by psychologists, in which the behavior of an animal undergoes modification based on the ensuing reinforcement, Mendel and McLaren [90] labeled them *reinforcement learning* tasks. Reinforcement learning has been studied extensively by Barto and his co-workers [24, 21, 23, 136, 18, 15, 19, 7] and others [43, 153, 154], and they have

[1]The terms "indirect" and "direct" are used differently in the adaptive control literature (e.g., Goodwin and Sin [45], Narendra [102]) to differentiate between control design methodologies: Indirect methods obtain controller parameters through appropriate analysis of a model of the controlled process that is constructed on-line via explicit system identification; direct methods, on the other hand, estimate the controller parameters without constructing an explicit process model. As we discuss in Chapter 2, our usage of the terms direct and indirect is also somewhat different from that of other researchers in AI and connectionist learning. Barto and Singh [20], for example, define indirect methods as those involving explicit modeling of the process and not of the IP. Instead of restricting ourselves to either definition, we shall use these terms in the fairly broad sense defined above, which, in our opinion, encompasses various usages, all of which are closely related.

developed several direct reinforcement learning methods. A common characteristic of these methods is that they obtain training information by actively perturbing the learning system's outputs.

Building on this previous work, our own research is directed towards increasing the applicability and efficiency of direct reinforcement learning methods. Of the several questions that arise from research in applying learning methods to control problems, we are primarily concerned in this dissertation with the following: How useful are direct reinforcement learning methods for training controllers? Although direct methods are often perceived as being weaker, or more primitive, than indirect methods, a major point made in this dissertation is that they can yield faster and more reliable learning in certain types of control problems. We also present several algorithms, architectures, and training methods designed to increase the power of direct methods for learning control. Although these techniques are described in this dissertation using the framework of control problems, they are generally applicable to other kinds of learning problems as well.

As indicated by the above discussion, our research addresses issues in applying direct reinforcement learning methods to control problems. Therefore, we now present a brief account of learning control followed by a discussion of some of the central issues relevant to reinforcement learning control.

1.1 Learning control

To control a process, the controller has to manipulate process inputs to achieve some predefined objective. Historically, controllers have been designed by trial and error: the controller is modified iteratively until the performance specifications are met to satisfaction. Starting in the early nineteenth century, control theorists have been developing mathematically sophisticated methodologies for designing controllers for certain well-defined classes of control problems. For example, proven design techniques exist for control problems involving linear processes and certain types of objective functions, such as quadratic cost functions [86]. Using these techniques, controllers can be designed that are provably stable, have desired response characteristics, or perform optimally. However, despite substantial efforts, similar techniques are not yet available for more general classes of control problems involving processes that are nonlinear and stochastic. Uncertainty and noise further complicate the problem of controller design. Faced with the formidable conceptual and analytical difficulties in solving such control problems, one alternative has been to incorporate adaptive and learning methods into the controller [45].

If a controller designed using conventional techniques performs poorly, it is usually due to uncertainty about the controlled process. Conventional control design techniques produce controllers based on mathematical models of the processes to be controlled. This approach is based on the assumption that the process model accurately replicates all relevant aspects the behavior of the process, and hence a controller that can achieve the control objectives with the process model should be able to perform well with the real process. The performance of a conventional controller therefore depends critically on the accuracy of the process model. Unfortunately, for many real-world systems, models are based on heuristics and the designer's experience. Sometimes the controller is directly implemented without reference to an explicit process model. Again, the designer relies on heuristics to determine the controller's behavior.

Although with conservative control objectives, conventional design techniques can yield controllers that perform satisfactorily in practice, researchers have also developed various methods for overcoming the drawbacks of these techniques. The goal of adaptive control [45] is to improve control performance by reliably estimating unknown parameter values of a partially specified process model or controller. This estimation is often performed *on-line*, i.e., as the process evolves under the influence of the controller. Hence, an adaptive controller extracts relevant information from its interactions with the process and uses this information to improve control performance over time. Moreover, adaptive controllers can track changing process characteristics by retuning the parameters. Clearly, as the complexity of the controlled process increases, it becomes more difficult for the designer to determine the process model or controller parameters, making adaptive methods more useful.

These same advantages are also exhibited by methods for learning control. The distinction commonly made in the literature between adaptive and learning control is that in addition to parameter estimation, learning controllers perform advanced decision making, pattern recognition, and other functions resembling human behavior [42]. Learning controllers can automatically select appropriate control behavior for different modes of operation of the process without re-adaptation—provided they have had earlier exposure to these modes of operation—while adaptive controllers must re-adapt. In other words, controllers can learn to recognize previously-experienced situations and select the appropriate control behavior they had stored in a "long-term" memory. This implies that in addition to learning on-line like adaptive controllers, learning controllers can also be trained *off-line* over a wide range of operating conditions, leading to good overall performance when they are brought on-line.

Another motivation for using learning control methods is the potential they offer for coping with disturbances such as friction and gear back-lash that are not easily represented by process models. By compensating for these unmodeled factors when generating control actions, learning control methods can improve performance substantially. Finally, investigating how control can be learned using artificial learning systems can shed light on how humans and animals learn motor skills. Conversely, detailed studies of the acquisition of motor skills by animals can yield useful insight and guidance for the design of artificial systems for learning control. With the advent of several novel learning methods, the distinction between adaptive and learning control has blurred substantially, and many today regard the distinction between the two as insignificant. We therefore use only the more comprehensive term "learning control" in the sequel.

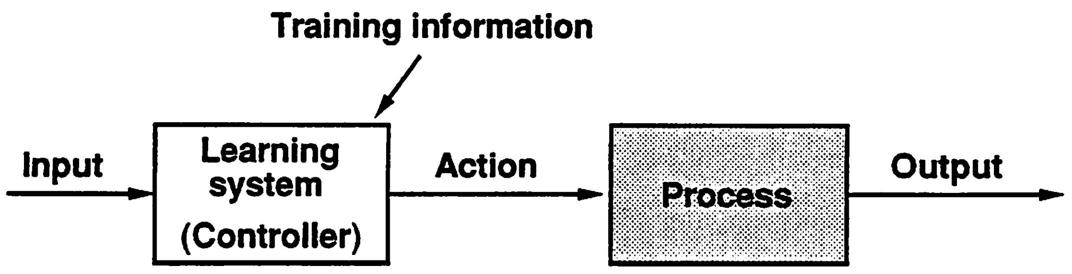


Figure 1.1. *The basic problem setup for learning control. The controller is to learn to control the process through the generation of appropriate control actions. It has to do this using the information supplied to it in the task specification, possible feedback of current and past process outputs and past control actions, and training information.*

Let us now consider learning control tasks in greater detail. In our rather general definition of a learning control task,[2] the learning system plays the role of a controller whose actions serve as control inputs to a process as shown in Figure 1.1.[3] The controller generates control actions based on its inputs, which include the task specification and possible feedback of current and past process outputs and past control actions. The output of the process is an observation of the process state obtained using a set of sensors. Guided by some form of training information, the controller has to learn to generate appropriate control actions for performing the specified task. In general, the training information pertains to the performance of the controller as measured by some predefined *index of performance* (IP).

The learning process involves repeated invocation of three consecutive functions: measurement, decision, and modification. *Measurement* is used to obtain information about the value of the IP and, more generally, about the control task. This information is used in the *decision* process to decide what, if any, adjustments are necessary to the controller in order to improve control performance. Finally, appropriate *modification* of the controller's behavior is effected based on the measurement and decision processes.

[2]Arimoto et al. [8], for example, give a more restrictive definition used mainly by control theorists.

[3]In all the block diagrams in this dissertation, the learning system modules are drawn using white boxes and modules external to the learning system are drawn using shaded boxes.

As discussed by Barto [17] and others, the degree to which the available training information is informative about the IP critically influences the controller's training. If the training information specifies the control actions that would result in optimizing the IP, as is the case in *supervised learning tasks*, or tasks involving *learning from examples*, the learning problem is relatively straightforward. The decision function becomes superfluous in such learning tasks, and the emphasis is on modifying the controller's behavior based on the desired control actions obtained through "measurement." If, however, the available training information does not directly specify the desired control actions—as is usually the case in most control problems—learning control becomes more complicated: a decision process for inferring control actions that improve performance as measured by the IP becomes necessary. Both direct and indirect learning methods described above are applicable in such circumstances. Our focus in this dissertation is on assessing the utility of direct reinforcement learning methods in such control tasks. In the next section, we outline several issues relevant to reinforcement learning control that are addressed in this dissertation.

1.2 Reinforcement learning and control

In the foregoing discussion, we distinguished between direct and indirect methods for learning control. A major contribution of this dissertation is an examination of the relative merits of these two classes of learning control methods. Empirical results of training controllers on various control problems are presented that enable one to compare and contrast the performance of direct learning methods with that of indirect learning methods applied to the same control problem. These results illustrate the power and utility of direct reinforcement learning methods for learning control. Furthermore, the comparisons provide insight into the underlying characteristics of control problems that make application of one type or the other of these learning methods more effective. Related issues are discussed by Barto and Singh [20].

An important issue in reinforcement learning control is the difficulty in scaling direct methods to more complex reinforcement learning problems. The first problem, called the structural credit assignment problem, arises when the learning system's actions are the result of the actions of multiple components. In such cases, the contributing components must be assigned appropriate credit or blame for the observed value of the IP. We present a novel modular architecture that can be used for structural credit assignment in a reinforcement learning system. Related to the indirect methods described above, this architecture represents a compromise between modeling of the IP as required by indirect methods and the complete absence of models in direct methods.

The problem of using reinforcement learning to train controllers to perform more complex control tasks is also addressed in this dissertation. We have already remarked on the resemblance between reinforcement learning and the learning behavior exhibited by animals. By implementing "shaping," a technique used in experimental psychology for training animals, we demonstrate how learning to solve simpler problems can facilitate learning to solve more complex problems. Related work along these lines has been presented by several researchers (e.g., [125, 48, 151]).

Shaping can been implemented in two ways. First, the behavior of a controller can be shaped over time by gradually increasing the complexity of the control task as the controller learns. Second, shaping can be implemented *structurally* by using a multi-level architecture trained bottom-up so that previously learned control behaviors are used as primitives for learning more complex behavior. Both of these shaping methods are illustrated by concrete examples in this dissertation. We also present a multi-level controller design incorporating several novel ideas that considerably improve learning efficiency.

An important caveat must be mentioned here. Our approach to learning control has been almost entirely experimental in nature, and we have not attempted to subject any controller presented in this dissertation to the rigorous analysis customary in traditional control theory. Furthermore, by resorting to heuristics, we have been forced to abandon orthodox control design practices such as ensuring stability of the controlled process and convergence of the design method. Thus, we are only dealing with some of the complexity of control as developed by control theorists. As a result of relaxing these constraints, however, we are able to address more complex control problems (e.g., problems involving nonlinear processes) for which no orthodox solutions exist.

Also note that the issues in learning control described above are important independently of the learning paradigm used to implement the controller. However, because we have used connectionist learning methods to implement all the controllers in this dissertation, we provide a brief overview of connectionist learning systems.

1.3 Connectionist learning systems

Connectionist networks, also known as artificial neural networks, have generated great interest because of their computational properties. Several different types of connectionist networks can serve as mechanisms useful for approximating various functions that are relevant to learning control. Research on connectionist learning methods is growing rapidly and rather than attempt a complete review here, we restrict ourselves to providing some references. An extensive overview of connectionist learning is provided by the three volumes edited by Rumelhart and McClelland [117, 89] or the volume by Hecht-Nielsen [57]. A good elementary introduction is provided by Lippmann [83]. Hinton [58] reviews a wide range of connectionist learning methods. Connectionist approaches to learning control have received significant attention in recent years. Barto [16] provides an overview of various approaches to learning control using connectionist networks. His emphasis is on the relationship between the research in connectionist learning methods and more traditional research in control and signal processing. Additional reviews are provided by Mars [87] and Werbos [144].

The basic components of a connectionist learning system are simple, neuron-like processing elements called *units*. Units receive inputs through links connecting them to other units or to the environment. Each input link to a unit has a *weight* associated with it, and the *activation level* of the unit, which determines its output, is often computed as the weighted sum of the inputs. The entire system of interconnected units is called a *network* and is presumed to be operating in an *environment*. Specific units in a network are designated as either *input units* or *output units*; these serve as sites of interaction between the network and its environment. All other units are called *hidden units*.

A network is often classified by the connectivity pattern between its units, which can be represented as a directed graph with nodes corresponding to the units and links corresponding to connections from one unit to another. For example, the directed graph associated with a *feed-forward* network is acyclic, whereas that associated with a *recurrent* network has cycles. Computation of a network's output for a given input is governed by a propagation rule that depends on the structure of the network. For example, in feed-forward networks, the activations of the input units are set by the environment, and successive layers of units then compute their outputs. The exact process by which the units determine their activations and outputs varies according to the type of unit.

Learning is achieved in a connectionist network by adjusting the weights of units in the network so as to compute a desired function from inputs to outputs. Several learning algorithms have been developed for training various types of connectionist units on different kinds of learning tasks. For example, many of the controllers for the various examples in this dissertation were trained using a direct reinforcement learning algorithm we devised called the stochastic real-valued unit (SRV) algorithm (Chapter 3 and [48]). The utility of SRV units is manifest when training connectionist networks to become controllers because most control tasks require the use of real-valued control signals rather than binary signals. We also present some new theoretical results obtained for the SRV algorithm. These include a convergence theorem that applies to a simplified version of the SRV algorithm. Detailed descriptions of other learning algorithms used in this dissertation can be found in the references cited above and in Appendix B.

1.4 Organization of the dissertation

The relative merits of direct and indirect methods for learning control are discussed in Chapter 2, where we elaborate the computational tradeoffs between the acquisition of knowledge and its use for control implicit in each method. Some of the main issues in learning control are also discussed in that chapter. A detailed description of the SRV algorithm, which is an example of a direct reinforcement learning algorithm, is presented in Chapter 3, followed by some associated theoretical results. Next, in Chapter 4, several examples are presented to illustrate the power and utility of direct reinforcement learning techniques for learning control, and the performance of the direct learning methods is compared with that of indirect learning methods applied to the same control problems. Chapters 5 and 6 present architectures and training procedures that enable scaling reinforcement learning to more complex problems. In Chapter 5, a novel modular architecture is presented that can be used for structural credit assignment in a reinforcement learning system. Methods of implementing shaping are described and illustrated using examples in Chapter 6. Finally, some conclusions based on the results presented in the dissertation are discussed in Chapter 7, where we also suggest directions for future work.

CHAPTER 2

ISSUES IN LEARNING TO SOLVE CONTROL PROBLEMS

This chapter begins with a review of some of the salient issues pertaining to learning control. In this review, particular emphasis is placed on the bases for choosing between direct and indirect learning control methods. Problems specifically associated with direct reinforcement learning methods are discussed in some detail.

2.1 Learning to solve control problems

In the general problem of learning control, the learning system plays the role of a controller that selects actions, y, from a some set of possible actions, Y, to serve as control inputs to a process, as depicted in Figure 2.1. The output of the process, z, is the process state, or, more realistically, an observation of the process state obtained using a set of sensors. Guided by the training information supplied to it, the controller has to learn to generate appropriate control actions for performing the task specified by its input, x. We emphasize the following important observation with regard to the depiction of controllers in this dissertation. Due to the absence of any feedback paths in Figure 2.1 and other figures in this dissertation, the controllers might appear to be restricted to what control theorists call *open-loop* or *feed-forward* controllers. We stress that this is by no means the case. In addition to the task specification, the input, x, to the controller can also include feedback of current and previous process outputs as well as previous control actions, thereby permitting closed-loop control.

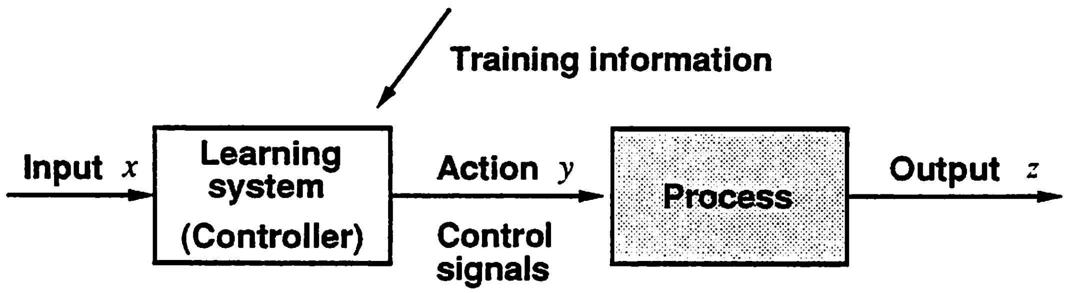


Figure 2.1. *The basic problem setup for learning control. The controller is to learn to control the process through the generation of appropriate control signals. It has to do this using the information supplied to it through the task specification, possible feedback of current and past process outputs and past control actions, and training information.*

Whether the controller is open-loop or closed-loop depends on the definition of the set X of controller inputs. Feedback paths have been omitted in the figures to include the possibility of open-loop control.

As described in the previous chapter, learning control involves modifying the controller's behavior to improve control performance as measured by some predefined index of performance (IP). The controller implements a control function $F_W : X \to Y$, with the subscript W denoting the parameters of the controller that determine what function is computed. The term "parameters" is being used loosely here; W denotes the memory buffer used in rote learning, the rules in a rule-based learning system, the decision tree of a controller using decision tree methods such as ID3 [112], the weights in a connectionist network, or the parameters in a more conventional adaptive control scheme. Learning appropriate control behavior involves determining W so that the resulting control function, F_W, has the desired performance as measured by the IP.

Three main questions must be answered when designing a learning controller:

(1) What kind of training information will be available to the controller?

(2) How should the learned control rules be represented?

(3) How should the inputs to the controller and its actions (the control signals) be represented?

Answering all of these questions is essential in designing a learning controller. However, this chapter focuses on how the form and content of the training information available in various kinds of control problems affects controller design and performance. We do not address questions about the controller's input and output representation and about the details of the learning algorithms; readers interested in these questions are referred to Barto [16, 17].

2.2 Training information

The measurement step of the measurement-decision-modification learning cycle described in the previous chapter is used to obtain information about the controller's performance and about the control problem in general. It is usually assumed that there is some kind of a "teacher" in the environment that supplies performance information to the controller.[1] Several researchers have suggested that learning tasks can be classified based on the quality of the training information provided by the teacher (e.g., [17, 68]). One can distinguish two major classes of tasks based on this criterion: *supervised learning tasks* and *reinforcement learning tasks*. In supervised learning tasks, the teacher provides target actions or error gradient vectors that specify how the controller should modify its actions so as to improve performance (Figure 2.2a). Thus the teacher can be seen as instructing the controller about the actions to execute in order to improve performance.

In contrast, the role of the teacher in reinforcement learning tasks is more evaluative than instructional, and the teacher is sometimes called a *critic* because of this role. As depicted in Figure 2.2c, the critic provides evaluations of consequences of the controller's actions, leaving it to the controller to determine how to modify its actions so as to obtain better evaluations in the future. Often, the critic's evaluation is the value of the IP that has to be optimized, although it is possible that the critic's evaluation function differs from the actual IP value in some kinds of learning

[1]Except in the case of *unsupervised learning* tasks, in which the necessary information can be considered to be built into the learning system (see [13, 17]).

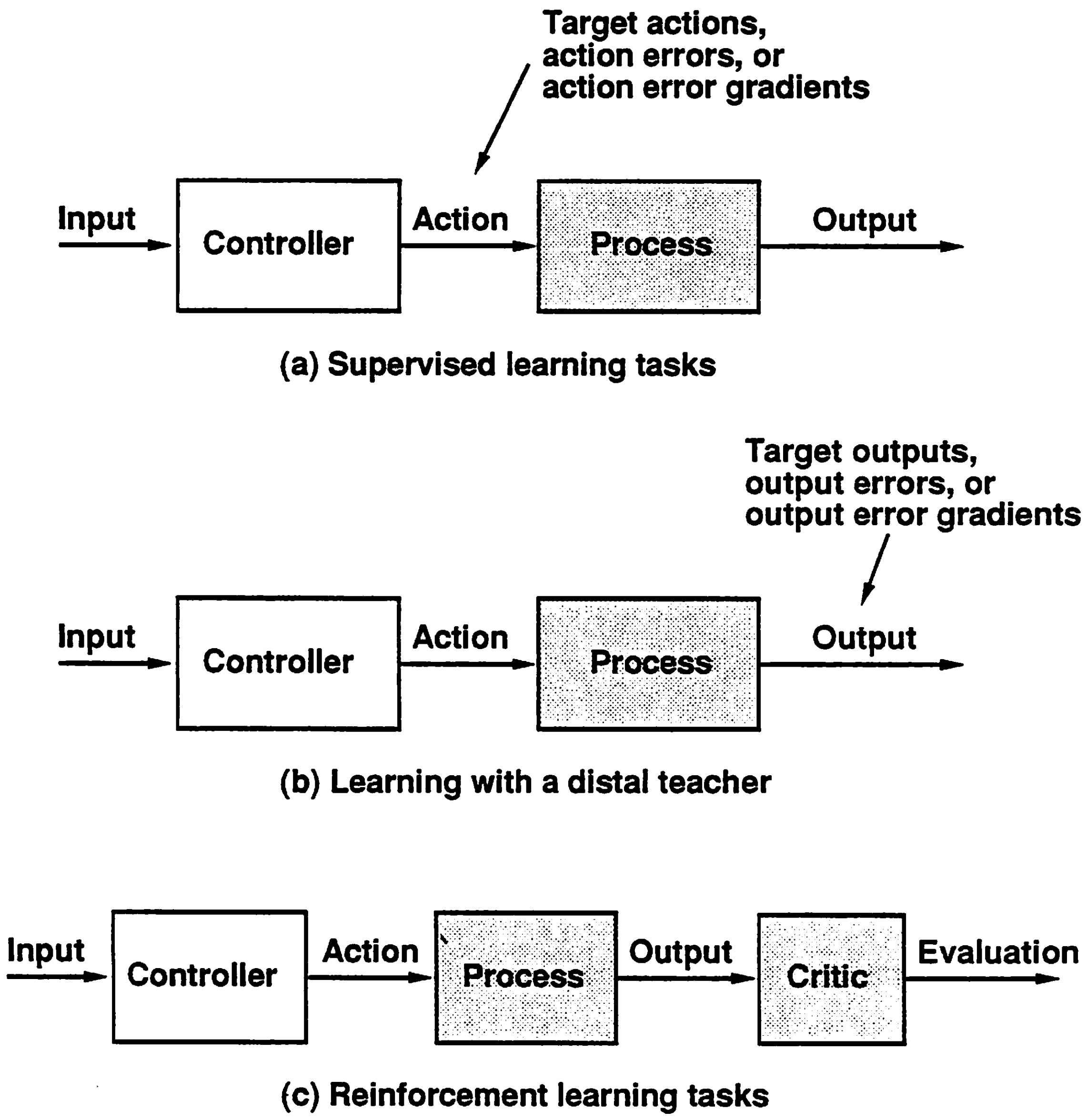


Figure 2.2. *A schematic of the three classes of learning tasks described in the text. These are (a) Supervised learning tasks, (b) Tasks involving learning with a distal teacher, and (c) Reinforcement learning tasks.*

tasks. The critic therefore provides the controller with information regarding the appropriateness of the control behavior of the current control function, F_W. To provide this information, the critic has to know the characteristics of the desired control behavior, but it need not explicitly know a control function that has the desired behavior. From the above description, the main difference between the two classes of learning tasks is that the critic in reinforcement learning does not explicitly tell the controller *what to do* to improve performance (as measured by the IP), whereas the teacher in supervised learning does.

An intermediate class of tasks that shares aspects of both supervised and reinforcement learning tasks involves what Jordan and Rumelhart [68] call *learning with a distal teacher*. The teacher in these learning tasks is termed "distal," meaning "far from the point of origin," because the training information it provides pertains to the outputs of the unknown controlled process instead of to the actions of the controller (Figure 2.2b). As indicated in the figure, the teacher might provide distal targets, distal output errors, or distal error gradients. Distal training information is very common in control because many control problems are formulated either as regulation problems, in which the process output has to be maintained at a specified set-point, or as tracking problems, in which the process output has to follow a reference trajectory. Consequently, control theorists, including adaptive control theorists, have devoted considerable attention to control problems involving distal teachers.

Inasmuch as the teacher provides targets, errors, or error gradients, albeit in distal coordinates, tasks involving learning with a distal teacher are similar to supervised learning tasks. At the same time, as in reinforcement learning tasks, the controller has to discover the right control actions, i.e., the control actions that drive the process outputs to the target outputs. Note that alternatively one can view the process and the critic together as a composite "process" whose "output" is the evaluation. Thus regarded, reinforcement learning tasks are special cases of learning with a distal teacher with the objective of extremizing the distal output. In taking this perspective, the distal "error" or the distal "error gradient" in reinforcement learning tasks can be assumed to be any positive value if the evaluation is to be maximized and any negative value if the evaluation is to be minimized.

2.3 Learning methods

Learning *methods* used for the three classes of learning control tasks delineated above reflect differences in the content of the training information. Let us first consider control problems defined as supervised learning tasks. Because desired actions or action error gradients are available in supervised learning tasks, adjusting the controller parameters, W, to reduce action errors is relatively straightforward. Considerably sophisticated supervised learning methods are required, however, to ensure that the control function, F_W, exhibits interpolation and extrapolation properties that imply good generalization of control. Several researchers have used supervised learning methods to train controllers in supervised learning tasks (see, for example, [150, 53]). These researchers have used a preexisting expert as a source for the training data. The (usually human) expert supplies a sufficiently large set of training pairs specifying the desired actions for various controller inputs, and the controller is trained to produce the corresponding action for each input. One of the earliest examples of the use of this method is the training of industrial robots to perform repetitive operations in an assembly line. For example, controllers are "taught" to paint car body parts by a human who moves the paint sprayer through a specific trajectory. Points from the trajectory are sampled and stored for later use in trajectory generation. This technique is very easy to implement and is widely used in industrial robotics today.

Widrow and Smith [150] used data provided by a human expert to train a controller to solve a pole-balancing problem. Widrow draws the parallel between this approach and knowledge acquisition from an expert when building an expert system. Once trained, the controller can match the expert's performance or even improve upon it. Handelman et al. [53], for example, show how a connectionist controller can use crude training information from a hand-crafted rule-based controller to learn a generalized control function that has improved performance. However, such generalized control functions may not always lead to improved performance, especially when the teacher provides unreliable training data. Moreover, most complex tasks cannot be performed satisfactorily by any existing controller, and hence reliable training data is difficult to obtain. Most learning control problems therefore involve either reinforcement learning or learning with a distal teacher.

Methods for solving learning control problems involving reinforcement learning and learning with a distal teacher are more complex than methods applicable to supervised learning tasks. As mentioned in Chapter 1, the decision-making part of the measurement–decision–modification learning cycle becomes crucial in problems involving reinforcement learning or learning with a distal teacher because appropriate control actions must be *inferred* from evaluations or distal training information. In other words, a method for bridging the gap between the form in which training information is available to the controller (evaluations, distal targets, etc.) and the form of information required for successful control (appropriate control actions) is necessary. Methods devised by researchers for bridging this gap fall into two major categories. *Indirect*[2] methods involve constructing a model of the transformation from the controller's actions to evaluations or distal targets and using the model to obtain training information for the controller. On the other hand, *direct*, or model-free, methods obtain the requisite training information by perturbing the process and observing the effect on the evaluations or the distal process outputs. A more detailed description of these methods follows.

[2]As mentioned in Chapter 1 (see footnote on page 2), our usage of the terms direct and indirect differs somewhat from other usages of these terms.

2.3.1 Indirect methods

Indirect methods can use models in at least three different ways. In conventional indirect adaptive control, a parametrized process model is used as a mathematical representation of the process from which an appropriate control law can be obtained analytically. The process model's parameters are adapted on-line through an operation commonly known as *system identification* in the control literature. Because the control law is derived analytically using the current model, indirect adaptive control methods differ significantly from learning control methods, which use the model to obtain information for *training* the controller.

An indirect method can also use a model of the process in the "forward" direction to simulate the process behavior over time. This is the approach used most often in AI game-playing programs (e.g., [120, 14]) in which a model of the game is used to generate search trees. Many heuristic search algorithms have been developed in AI [14] for this kind of search. Clearly, these heuristic search algorithms can also be applied to control problems other than game-playing and in situations in which the model has to be constructed on-line. The main drawback of this approach is that the forward search process is, in general, underconstrained and hence computationally expensive.

In contrast, a more constrained indirect method is to use the model in an "explanation" mode to infer appropriate control actions from the control objectives. The basic idea underlying this method is the following. To determine the appropriate control actions, it is necessary to determine how to change the control actions so as to effect the desired changes in the distal process output or the evaluation. One way to do this is to use information regarding the gradient of the distal output or evaluation with respect to the controller's current action. This gradient can be obtained if one has a model of the process (or of the process and the critic) in a form that can be differentiated. The model's Jacobian[3], evaluated at the model's current input, can then be used to transform desired changes in the outputs of the model into changes in the inputs to the model. This is illustrated in Figure 2.3.

An indirect method for obtaining gradient information was described in a connectionist setting by Jordan [64] (see also Jordan and Rumelhart [68]). Barto [16] calls this method "Differentiating a Model." Similar techniques described in the adaptive control literature are classified as *sensitivity modeling* techniques [103]. Note that although gradient computation is meaningful only for models that can be expressed as differentiable functions, it is easy to visualize an analogous operation when other kinds of models are used. For example, backward-chaining techniques [14] used in AI perform an analogous operation in rule-based systems. Nevertheless, we use the term "gradient computation" to refer to the operation performed by all techniques that use models in an "explanation" mode. In this dissertation, we are primarily concerned with gradient-based indirect methods, and hence we do not discuss other indirect methods any further. Hereafter, by an indirect method we mean a gradient-based indirect method unless we state otherwise.

Using a task involving learning with a distal teacher, Jordan and Rumelhart [68] show how a connectionist network can implement this technique in an elegant manner using back-propagation [80, 111, 116, 143] . Specifically, the network is trained to be what they call a "forward model" of the process via system identification. Training the network is a supervised learning task with the training data obtained from the process being modeled. Once the forward model has been trained to a sufficient degree of accuracy, it is held fixed and the controller's training is begun. In this stage, the forward model is used to predict the process output for each control action generated by the controller, and the difference between the desired process output

[3]The Jacobian of an input–output transformation is a local linear approximation to the transformation that relates small changes about a nominal input value to corresponding changes in the nominal output value.

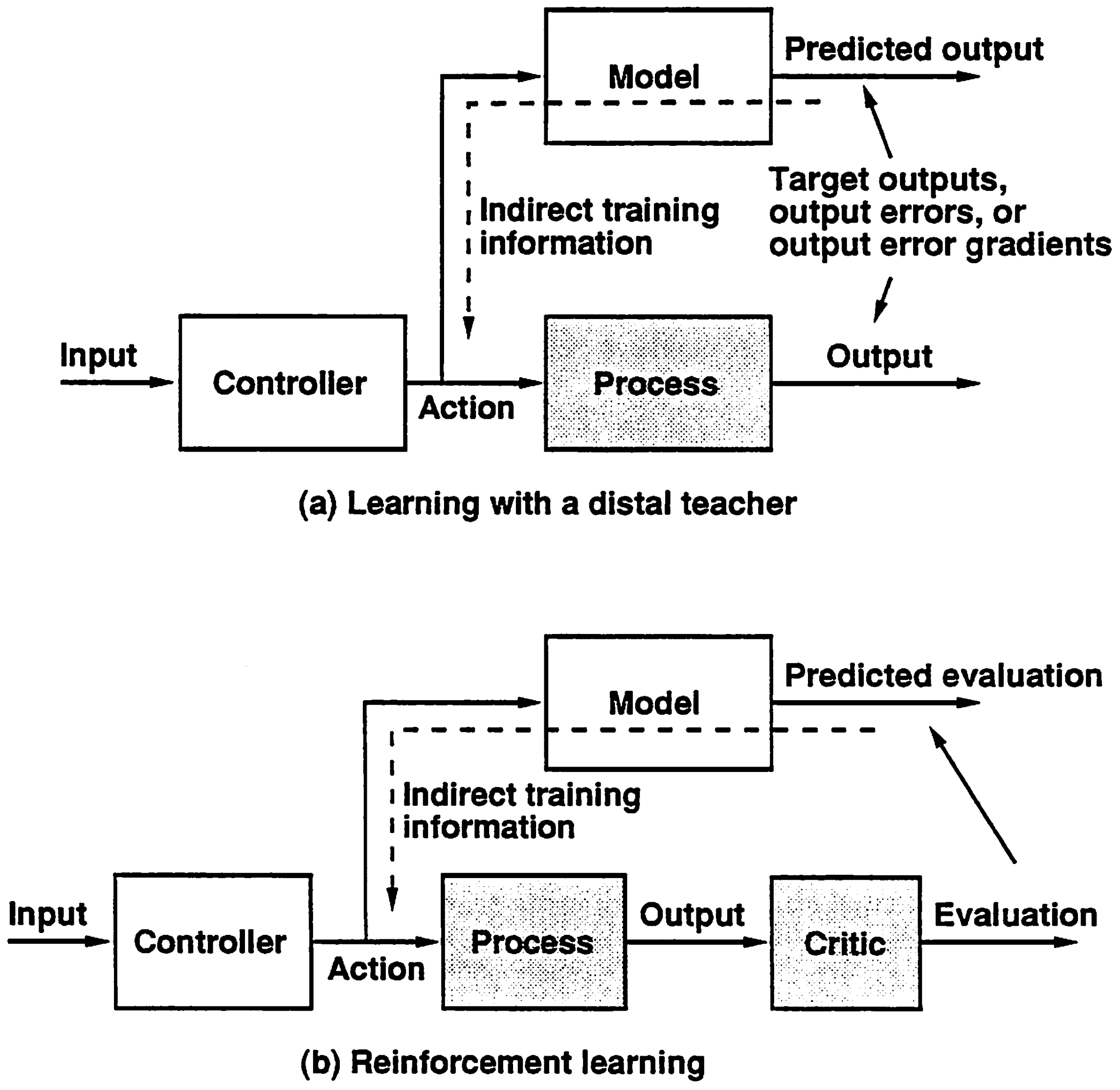

Model
Predicted output
Indirect training information
Target outputs, output errors, or output error gradients
Input
Controller
Action
Process
Output
(a) Learning with a distal teacher
Model
Predicted evaluation
Indirect training information
Input
Controller
Action
Process
Output
Critic
Evaluation
(b) Reinforcement learning

Figure 2.3. *A block diagram of the gradient-based indirect method for learning control. This method can be applied to control problems involving learning with a distal teacher or reinforcement learning. After Jordan and Rumelhart [68] and Barto [17].*

and the predicted output is back-propagated through the forward model, yielding the error gradient with respect to that control action.

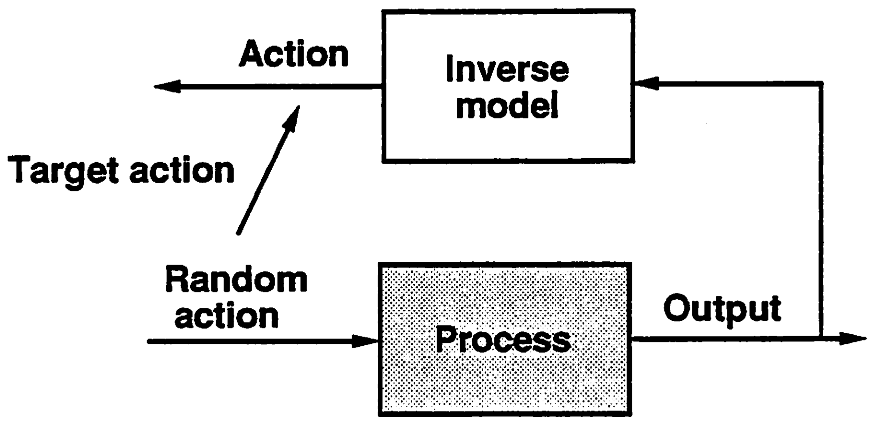
Jordan [64] shows that the back-propagation process computes a factorization of the transpose of the network Jacobian. Jordan also demonstrates the utility of this technique in a variety of control problems [65, 64, 68]. Variants of this technique have been applied to control problems by Kawato [70], Miyata [97], Nguyen and Widrow [107], and others. However, as noted by Barto [16], as well as Jordan and Rumelhart [68], the model need not be a back-propagation network for this technique to apply. Any forward model of the process that can be differentiated can be used. In their study of reaching behavior in monkeys, for example, Massone and Bizzi [88] use a Kinematic Network [101] model of a limb in place of a back-propagation network model.

A similar approach has been proposed for solving reinforcement learning tasks by Jordan and Jacobs [66], Munro [100], Werbos [145], and Williams [154]. As illustrated in Figure 2.3b, the idea here is to train a connectionist network to model the process together with the critic, instead of the process alone. Training inputs to the network are obtained by generating random control signals and observing the corresponding evaluations returned by the critic. Once the model network has been trained using an appropriate supervised learning method, it can provide the gradient of the evaluation with respect to the controller's actions by, for example, back-propagating through the model. Werbos [144] also suggests how this approach can be used with explicit models for both the process and the critic.

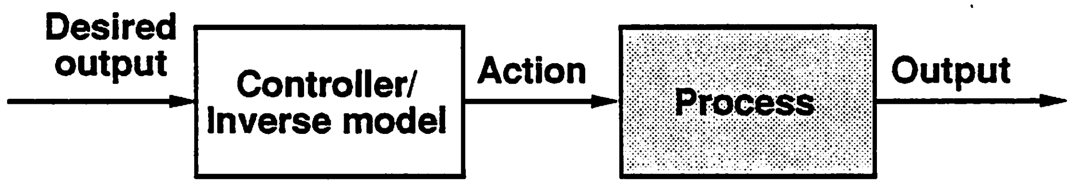
2.3.2 Direct methods

Instead of resorting to model construction, direct methods use the process itself as a source of training data for training the controller. For learning tasks with distal targets, if the command input to the controller is the desired output of the process, direct identification of an inverse model of the process has been proposed as the solution [149]. Such a method has been called "direct inverse modeling" [67] in connectionist learning literature and "input matching" in adaptive control literature [45]. Training data for the controller are obtained by feeding a variety of control signals to the process and observing the resultant process output. A supervised learning method is used to train the inverse model with the observed process output as input and the control signals as the desired actions, as shown in Figure 2.4. Once trained, the inverse model can be used as a controller that produces an appropriate control action for any desired process output. Widrow et al. [149] used this method to control linear systems. Other researchers have applied this method to the control of nonlinear thermodynamic systems [131] and for learning the inverse kinematics [10, 78] and inverse dynamics [71, 93] of robots.

Direct inverse modeling appears to be an attractive method for learning control because it is easily implemented. However, its applicability and usefulness depend on the characteristics of the controlled process. With dynamic processes, for example, the data obtained for training the inverse model depend on the initial state of the process, and hence the process state must be used as context for the inverse model in order to ensure effective control. Furthermore, as emphasized by Jordan [64], inverse identification may not yield good results when the process inverse is not unique, i.e., when many different control actions result in the same process output, and hence the training data can include different target actions for the same process output. In such cases, controllers trained using most supervised learning techniques tend to produce for each process output a control action that is in the convex closure of all the target actions used in training. Therefore, unless the inverse image of a desired process output is a convex set, the control action computed by the controller might not lie in the inverse image. Finally, Ydstie [156] discusses the advantages and disadvantages of using direct inverse modeling from a control systems perspective and considers stability issues that arise when attempting to obtain an inverse model for a process with unstable zeros.



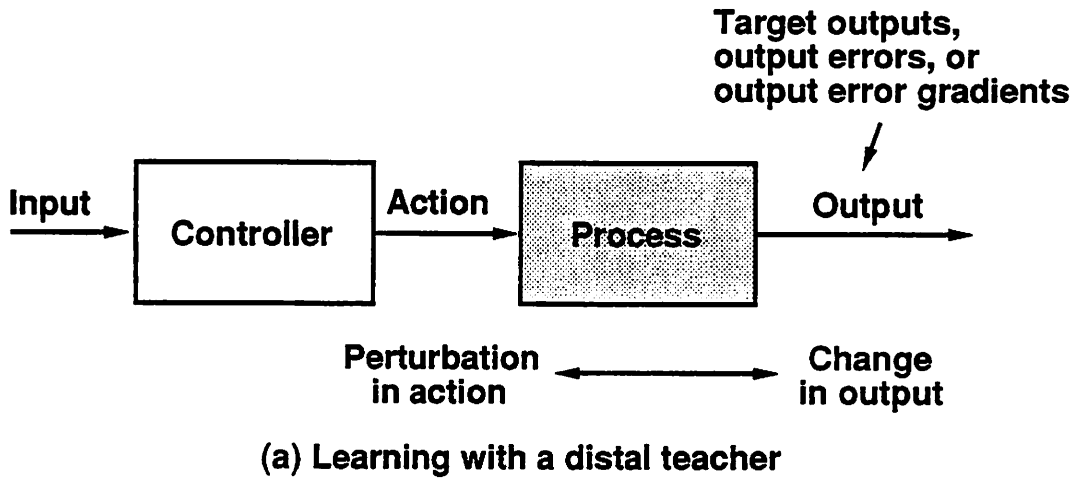
(a) Training the inverse model



(b) Using the inverse model as a controller

Figure 2.4. *Control acquisition through direct inverse modeling. Part (a) shows the configuration used for training the inverse model, while part (b) shows how the inverse model is used to control the process. See also Barto [16].*

We do not discuss direct inverse modeling further in this dissertation. Instead, we focus on another subclass of direct methods that can be considered as the direct analogues of gradient-based indirect methods. These are direct methods for determining the gradient of the evaluation or the distal output with respect to the controller's actions without resorting to model construction. These methods estimate gradients by actively perturbing the controller's actions and observing the consequent changes in the evaluation or the distal output, as depicted in Figure 2.5. Perturbation is usually accomplished by adding either sinusoids of known frequencies or random noise with known characteristics to the actions of the learning controller.


Target outputs, output errors, or output error gradients
Input
Controller
Action
Process
Output
Perturbation in action
Change in output
(a) Learning with a distal teacher

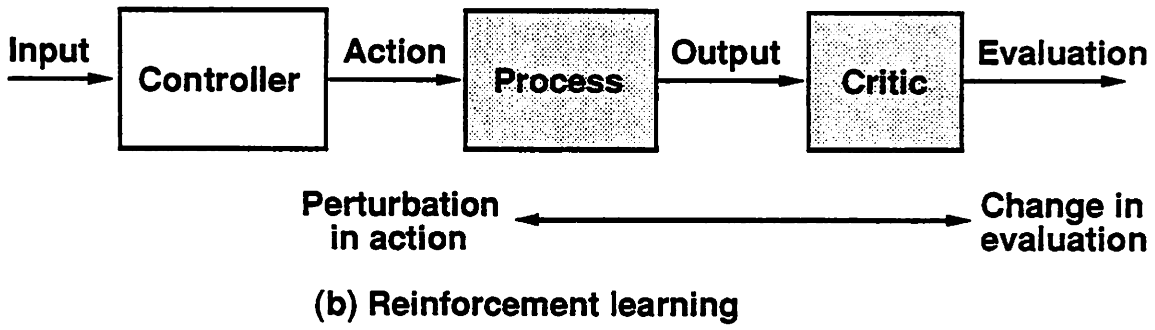


Figure 2.5. *A block diagram of the direct, or model-free, learning control method based on gradient estimation through perturbation. This method can be applied to control problems involving learning with a distal teacher or reinforcement learning.*

Gradient estimation through active perturbation is a very old idea and has seen implementation in the parameter perturbation approach [32] studied by adaptive control theorists, the Kiefer-Wolfowitz process [72] and other related stochastic approximation processes, and various learning algorithms studied by learning automata theorists [104] and psychologists [90]. As mentioned in Chapter 1, most direct reinforcement learning algorithms (e.g., [43, 24, 23, 136, 18, 154, 48]) are also based on this idea. We discuss direct reinforcement learning methods in detail in Chapter 3, where we also discuss how *associative* reinforcement learning methods, introduced by Barto, Sutton, and Brouwer [24], can be used to perform *context-dependent* action improvement via direct gradient estimation, thereby enabling the controller to associate different optimal actions with different inputs. The utility of associative reinforcement learning in control has been discussed by Barto, Sutton, and Anderson [23], who used it to control a pole-balancing system. Other examples of the use of this approach are seen in Barto, Sutton, and Brouwer [24], Franklin [40, 41], and Gullapalli [48, 49].

2.3.3 Comparing direct and indirect methods

Indirect methods for learning control are often perceived as having advantages over direct methods when a model of the process, or of the process and the critic, is available a priori. An accurate model provides the controller with substantial information regarding the process to be controlled. Moreover, a model provides means for incorporating domain knowledge useful to the controller. If an adequate model is not available, it can be constructed in various ways; in particular, it can be learned through interactions with the environment. Investment of computational resources in model construction is justified if using the model to obtain training information for the controller results in a net reduction in computational costs when compared with direct methods. The issue of local versus global models is of importance in this regard. Often, a local model, i.e., a model valid only in restricted regions of the process state and input spaces, is adequate for solving a given control problem. Construction of a local model is not only more practical than construction of a global model but also entails lower computational costs. On the other hand, a global process model can be useful for solving multiple control problems involving the same process, and therefore the cost of model construction can be amortized over a range of control problems. It must be noted, however, that uncertainty and noise can make constructing an adequate model, local or global, impractical in many real-world situations, and even if model construction were practical, possession of a model does not necessarily guarantee a solution to the control problem.

When an adequate model can be constructed, one still needs to decide if the model is to be constructed on-line during control, or off-line while control is not attempted. In the latter case, one has to take into consideration the relationship between the accuracy of the model and its utility for control. Two factors govern this relationship. First, the computational costs of constructing an accurate model can increase dramatically with the complexity of the process. Second, different control problems require varying degrees of model accuracy. For some problems, a fairly inaccurate model suffices, whereas for others a very accurate model is necessary. Unfortunately, one cannot usually tell beforehand how accurate a model is required to quickly acquire good control. Also because the range of control signals that will be used (or even required) for the control problem are not known a priori, a model constructed off-line must be more global than it would need to be if constructed on-line. Off-line model construction therefore suffers from the drawbacks of global modeling mentioned above. Moreover, one cannot always wait until an accurate model is constructed before control begins. Some form of control is usually desired while the model is being constructed. Finally, it is possible that the process being modeled is nonstationary, in which case the model has to be updated over time.

A potential solution to these problems is to construct the model on-line during control. This approach is often used in adaptive control (see Goodwin and Sin [45]). Because modeling and control are interleaved, the model is localized to the region of state space in which the controller maintains the process. However, with on-line modeling, control accuracy can suffer because the controller is initially trained using an inaccurate model. Therefore, it is usually more expedient to start on-line modeling with a model that has already been partially formed, especially in situations involving non-linear processes with complex parametrized models.

A second solution to the above problems with model construction is to avoid using a model if possible. Because they do not entail the costs of model construction, direct methods have the advantages of computational simplicity and no delay in training the controller. Moreover, because they use the process itself to obtain training information, direct methods might be able to perform well even when the process is nonstationary. Furthermore, not having to model the complexity of the process/process+critic might be a significant advantage for direct methods in some situations. A potential disadvantage of direct methods, however, is that the estimates of process behavior obtained through active perturbation can be less reliable than those obtained from an accurate process model. In such situations, learning control using direct methods might be slower than using indirect methods. Moreover, with direct methods there is no model available for reuse in other control problems.

From the above discussion, it is clear that both direct and indirect methods have advantages and disadvantages and that neither method is universally applicable. In particular, for some control problems, it is possible that the complexity of constructing a model exceeds that of solving the original control problem via direct methods, making direct methods preferable. Examples of such problems are provided in Chapter 4, along with experimental results that support using direct reinforcement learning methods. Barto and Singh [20] report related experiments in which they compare the "computational economics" of using a direct and an indirect learning method for Markovian decision problems.

2.4 Problems specific to reinforcement learning

Two requirements must be met if direct reinforcement learning methods are to be used for learning control. The first requirement is a critic that can evaluate performance in a manner that is sufficiently informative about the actual IP to permit effective learning. While reinforcement learning tasks, by definition, assume the availability of a critic providing some kind of evaluative performance feedback, it is not necessarily the case that each output of the critic is sufficiently informative about the problem's IP. Evaluations may occur sparsely over time, they may be corrupted by noise, and the evaluation criteria used might be non-stationary. Under these conditions, one approach is to have the controller construct a more useful internal critic that can facilitate learning. An example of a control task in which an internal critic is constructed through learning is presented in Section 4.3. Although the construction of more informative critics is a major area of current research [137, 25, 145], we do not discuss it in any detail in this dissertation. Instead, we note that this research is completely compatible with our own work, which is directed towards *using* such critics in learning systems.

Assuming that an appropriate critic is available, or that an adequate internal critic can be constructed, the second requirement for implementing direct reinforcement learning controllers is to find a reinforcement learning method that is compatible with the control problem at hand. In doing so, one must consider issues pertaining to the selection of representations for the controller's inputs and actions, the manner in which the search for better actions is conducted, and the decision process determining which actions are better than others. Chapter 3 presents an overview of approaches taken to meet these requirements. In that chapter, we also present a novel reinforcement learning algorithm called the SRV algorithm, which is an example of a method for learning via evaluative feedback from a critic.

Another issue that needs to be addressed is the issue of *scaling*. Scaling direct reinforcement learning methods to more complex tasks is hampered by the paucity of information in the evaluation signal used in reinforcement learning tasks compared with the training information available in supervised learning tasks. A second problem affecting scaling is the structural credit assignment problem of ascribing credit or blame to various components of the learning system. This problem becomes increasingly significant as the dimensionality of the control action vector increases. Both of these problems limit the complexity of the control tasks that can be solved using direct reinforcement learning methods. Our approaches to these problems are part of the contributions of this dissertation. The issue of maintaining learning performance while increasing the number of control variables is addressed in Chapter 5, and as we shall see in Chapter 6, having only evaluations available as training information is not necessarily a serious drawback, provided that the critic selects evaluation criteria judiciously based on the performance of the controller and that the learning system is designed to take advantage of task decomposition where possible.

CHAPTER 3

REINFORCEMENT LEARNING OF REAL-VALUED FUNCTIONS

This chapter presents an algorithm, called the stochastic real-valued (SRV) unit algorithm, for learning real-valued functions via reinforcement feedback. The chapter begins with a brief description of the relevant aspects of the theories of stochastic learning automata, pattern classification, and stochastic approximation that form the background for the approach taken in designing the SRV algorithm. Next, the SRV algorithm is described, followed by presentation of some associated theoretical results, including a convergence theorem that applies to a simplified version of the algorithm. Finally, some simulation results are presented that illustrate the performance of both the original as well as the simplified algorithms.

3.1 Learning automata and pattern classification

The discussion of stochastic learning automata tasks and supervised learning pattern classification tasks in this section parallels that of Barto and Anandan in [18]. Simple learning automata operating in stochastic environments have gained attention as models of learning since the work of Tsetlin [140] and of psychologists studying mathematical learning theory (e.g., [26] and [11]). A good review of the theory of stochastic learning automata is provided by Narendra and Thathachar [104]. A stochastic learning automaton interacts with its environment by randomly selecting an action as output to the environment, which, in turn, produces a random evaluation of the action. This evaluation (also known as a "success signal", "payoff", "reward", or "reinforcement") is used by the automaton to update its action probabilities. Learning involves adjusting the action probabilities so as to increase the expectation of favorable evaluations for future actions. Learning automata can be further classified according to the kind of evaluation they receive from the environment. If the set of possible evaluations is binary (denoting success/failure), the automaton is called a P-model automaton. If the evaluation is drawn from a finite set of more than two values, the automaton is a Q-model automaton, and if the evaluation is drawn from an interval of the real line (usually [0,1]), the automaton is an S-model automaton. The environment is *stationary* if the probability distribution used to produce the evaluation is constant over time. Otherwise, it is *nonstationary*.

It is important to note that in the above description of a stochastic learning automaton, the only input to the automaton is the evaluation signal. If the environment is stationary, such an automaton can learn to select the optimal action given the distribution of the evaluation. If the environment is non-stationary, the automaton can continuously track the optimal action for the time-varying distribution of the evaluation. But if we consider the task of learning optimal actions in different situations using such an automaton, it is clear that the automaton needs to consider input other than the evaluation signal. Such *context input* [24] serves to indicate the current state of the automaton's environment. Because the probability distribution governing the evaluations may depend on the state of the environment, the automaton has to use the context input to select the preferred action appropriate for that context. In other words, the automaton has to learn an *associative mapping* from the context input to the preferred action for that context.

One class of tasks involving learning associative maps consists of supervised learning pattern classification tasks. Pattern classification systems have been the subject of intense study since the 1950s. A comprehensive overview of pattern classification techniques is provided by Duda and Hart [33]. Learning in a pattern classification system involves using pairs of patterns and their class-labels, provided as training input to the system, to develop a classification rule that assigns the correct class label for each pattern or, in general, minimizes the probability of misclassification. It is usually assumed that the training pairs are produced randomly by the environment in which the classification system operates. Formally, for an m-class problem, the class ω_i is assumed to occur with probability $P(\omega_i)$, $1 \leq i \leq m$, and a particular pattern vector $\mathbf{x}$ from the i^{th} class is assumed to occur with probability $p(\mathbf{x} \mid \omega_i)$. Therefore the pattern classification problem is reduced to computing the a posteriori probability

$$P(\omega_i \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \omega_i)P(\omega_i)}{\sum_{j=1}^{m} p(\mathbf{x} \mid \omega_j)P(\omega_j)} \tag{3.1}$$

for every i and selecting as the class label that ω_i for which $P(\omega_i \mid \mathbf{x})$ is maximum. Equivalently, one can compute the *discriminant functions*

$$d_i(\mathbf{x}) = p(\mathbf{x} \mid \omega_i)P(\omega_i) \tag{3.2}$$

and select the label of the class with the largest discriminant function value.

In one set of techniques for pattern classification, called *linear discriminant function* techniques [33], the discriminant functions are linear functions of the input pattern of the form

$$d_i(\mathbf{x}) = \boldsymbol{\theta}_i^{\mathsf{T}}\mathbf{x} + c_i, \tag{3.3}$$

where θ_i, $1 \leq i \leq m$, are weight vectors. For the two class case, this is equivalent to forming a single discriminant function

$$d(\mathbf{x}) = \boldsymbol{\theta}^{\mathsf{T}}\mathbf{x} + c, \tag{3.4}$$

so that $\mathbf{x}$ is assigned to ω_1 if $d(\mathbf{x})$ is > 0 and to ω_2 otherwise. Solving the classification problem now involves using the training data to find weights that minimize the probability of misclassification.

Given randomly generated training pairs, if we can define a suitable error functional that quantifies the classification error, we can apply known stochastic approximation methods [69] to this problem. These methods have been developed for finding a set of parameters $\boldsymbol{\theta}$ that minimize (or maximize) a criterion function $J(\boldsymbol{\theta})$ in situations where observations of the function values for any given setting of parameters are corrupted with noise. It is assumed that we can observe either the random variable $q(\boldsymbol{\theta})$ that satisfies $E[q(\boldsymbol{\theta}) \mid \boldsymbol{\theta}] = J(\boldsymbol{\theta})$ or the random variable $g(\boldsymbol{\theta})$ that satisfies the relation $E[g(\boldsymbol{\theta}) \mid \boldsymbol{\theta}] = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ (i.e., $g(\boldsymbol{\theta})$ is a noisy measurement of the gradient of the criterion function with respect to the parameters). When the gradient information $g(\boldsymbol{\theta})$ is available, the Robbins-Monro algorithm [113] or its generalizations can be applied to update the parameters $\boldsymbol{\theta}$ so that the criterion function is optimized. Otherwise, the Kiefer-Wolfowitz [72] algorithm, or other similar algorithms based on obtaining estimates of the gradient from the noisy observations of the function values, $q(\boldsymbol{\theta})$, can be applied.

3.2 Associative reinforcement learning

Associative reinforcement learning tasks defined by Barto and Anandan [18] combine aspects of stochastic learning automata tasks and supervised learning pattern classification tasks. In associative reinforcement learning tasks, the learning system interacts in a closed loop with its environment. At each time step, the environment provides the learning system with input $\mathbf{x}$ chosen from a set of input vectors, X. Using this input, the learning system selects a scalar output y from a set of permissible outputs, Y. Based on both $\mathbf{x}$ and y, the environment computes and returns an evaluation or "reinforcement", $r \in R$. Ideally, we would like the system to learn to respond to each input with the output that has the highest expected evaluation. In keeping with the earlier work on learning automata, Barto and Anandan defined associative reinforcement learning tasks as involving selection of one of a finite set of outputs (Y is a finite), and for which the evaluation is a binary-valued success/failure signal (i.e. $R = \{0,1\}$). Their interest in such tasks led to the development of the A_{R-P} (associative reward-penalty) algorithm, which they prove has a form of optimal performance for associative reinforcement learning tasks satisfying certain conditions.

The above conditions on the output and the evaluation, however, are rather restrictive for most applications and one would like to extend the definition of associative reinforcement learning tasks to permit continuous-valued outputs and evaluations. Barto and Jordan [19] present a version of the A_{R-P} algorithm that can handle the latter case, where the evaluation returned by the environment can take on bounded continuous values (the so-called S-model case [28]). But learning continuous outputs is more difficult, and existing algorithms cannot be easily extended to do so. For example, A_{R-P} units (as defined in [18]) compute their binary-valued outputs by adding noise to their activations and thresholding the sum. Such units could be easily modified to produce continuous outputs by omitting the thresholding. Unfortunately, in such units there would be no control over the amount of noise that is added to the activation, and hence they would continue to produce random output values regardless of the duration of training. We now present our own efforts towards the development of a suitable algorithm for associative reinforcement learning tasks with continuous outputs and evaluations.

We begin by extending Barto and Anandan's definition of associative reinforcement learning tasks to cases in which the output y can take on continuous values and the environmental evaluation r lies in the interval $[0,1]$. Stated formally, associative reinforcement learning tasks involve the following interaction between the environment and the learning system. At time step t the environment provides the learning system with some context vector $\mathbf{x}(t)$ selected from a set of vectors $X \subseteq \Re^n$, where $\Re$ is the set of real numbers. The learning system then produces a random output $y(t)$ selected according to some internal *conditional* probability distribution over some interval $Y \subseteq \Re$. This distribution is conditioned on the input $\mathbf{x}(t)$. The environment evaluates the output $y(t)$ in the context of the input

$\mathbf{x}(t)$ and sends to the learning system an evaluation signal $r(t) \in R = [0,1]$, with $r(t) = 1$ denoting the most desirable evaluation. This evaluation is determined according to some conditional probability distribution $H : R \times X \times Y \rightarrow [0,1]$, where $H(r \mid \mathbf{x}, y) = Pr\{r(t) \leq r \mid \mathbf{x}(t) = \mathbf{x}, y(t) = y\}$. The objective of the learning system is to learn to respond to each input pattern $\mathbf{x} \in X$ with the action $y^{\mathbf{x}} \in Y$ with probability 1, where $y^{\mathbf{x}}$ is such that $E(r \mid \mathbf{x}, y^{\mathbf{x}}) = \max_{y \in Y}\{E(r \mid \mathbf{x}, y)\}$.

It is clearly possible to reduce associative reinforcement learning tasks to other kinds of tasks by placing restrictions on various aspects of the task definition. For example, in the case of a single context vector ($\mid X \mid = 1$) and when Y is a finite set, these tasks become examples of stochastic learning automata tasks discussed above. Alternatively, associative reinforcement learning tasks can be reduced to supervised learning pattern classification tasks by restricting both the output and the evaluation to discrete values and defining the evaluation to be a deterministic function of the input alone. Specifically, the "evaluation" has to be the label of the class of vectors or patterns in the input space to which the input belongs.

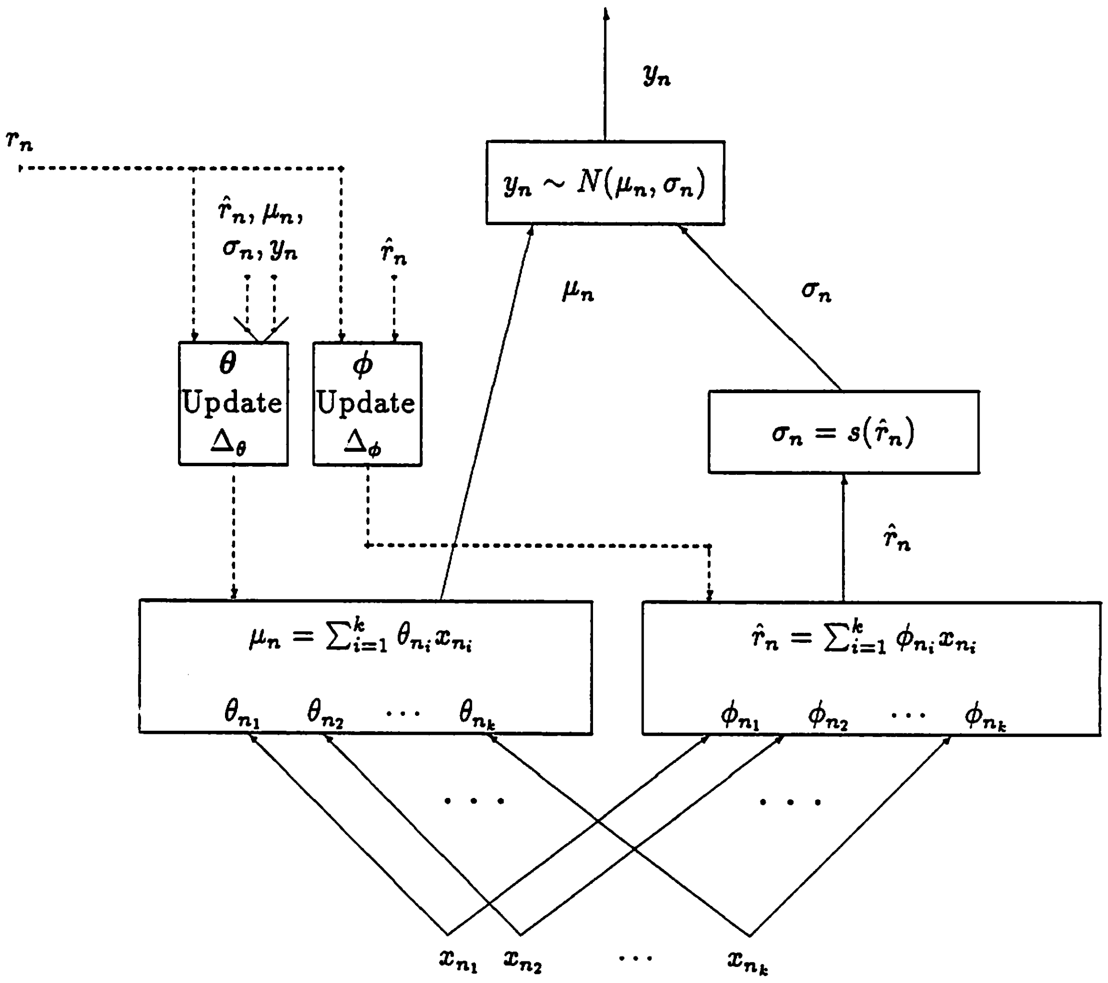
The *Stochastic Real-Valued* (SRV) unit algorithm [47, 48] incorporates the relevant techniques from the areas described above in a learning algorithm that (1) under certain conditions, learns the *real-valued* action that yields the highest evaluation in a given context (as a stochastic automaton does), (2) learns to associate different optimal actions with different contexts (as a pattern classifier associates different class labels with different input patterns), and (3) updates a vector of parameters, $\boldsymbol{\theta}$, specifying the optimal actions (as a stochastic approximation procedure does). The SRV algorithm is closely related to stochastic approximation procedures such as the Robbins-Monro [113] and the Kiefer-Wolfowitz [155] procedures and their generalizations [34]. An important feature of the SRV algorithm is that it can be implemented as a connectionist unit that can be incorporated into a network.

In this algorithm, the learning system computes its real-valued output as some function of a random activation generated using a Gaussian distribution. The activation at any time depends on the two parameters, the mean and the standard deviation, of the Gaussian distribution, which, in turn, depend on the current inputs to the unit. The SRV algorithm adjusts these two parameters so as to increase the probability that the algorithm produces the optimal real-valued output for each input pattern. The algorithm does this by maintaining the mean of its activation as an estimate of the optimal activation and by using the standard deviation to control the amount of search around the current mean value of the activation. A more formal description of the SRV algorithm is presented in the next section.

3.3 The stochastic real-valued (SRV) unit

Designed for associative reinforcement learning tasks defined in Section 3.2, the SRV unit has the structure shown in Figure 1. The interaction between the unit and the stochastic environment takes place as iterations of the following operations. (In the sequel, variables are subscripted by the iteration number at which their value is considered.) Iteration n begins with the unit receiving an input $\mathbf{x}_n \in X$. The unit uses $\mathbf{x}_n$ and two internal parameter vectors, $\boldsymbol{\theta}_n$ and $\boldsymbol{\phi}_n$, to compute the two parameters μ_n and σ_n of the Gaussian distribution used to generate the unit's output. The mean, μ_n, is the inner product $\boldsymbol{\theta}_n^{\mathsf{T}}\mathbf{x}_n$, and the standard deviation, σ_n, is computed in two stages by first computing an expected evaluation $\hat{r}_n$ as the inner product $\boldsymbol{\phi}_n^{\mathsf{T}}\mathbf{x}_n$ and then computing σ_n as a function s of $\hat{r}_n$. The function s is a monotonically decreasing, nonnegative function. Moreover, $s(1.0) = 0.0$, so that when the maximum reinforcement is expected, the standard deviation is zero. The output of the unit, y_n, is generated as a random variable depending on the Gaussian distribution with parameters μ_n, and σ_n. After the action is emitted, the unit receives evaluative feedback, $r(y_n, \mathbf{x}_n)$, from the environment, which it uses to adjust the parameter vectors $\boldsymbol{\theta}_n$ and $\boldsymbol{\phi}_n$.

Recall that the evaluative signal $r(y, \mathbf{x})$ is a random variable whose distribution coincides almost everywhere with the distribution $H(r \mid y, \mathbf{x})$. This distribution



y_n
r_n
$\hat{r}_n, \mu_n,$
σ_n, y_n
$\hat{r}_n$
$y_n \sim N(\mu_n, \sigma_n)$
μ_n
σ_n
θ Update Δ_θ
ϕ Update Δ_ϕ
$\sigma_n = s(\hat{r}_n)$
$\hat{r}_n$
$\mu_n = \sum_{i=1}^{k} \theta_{n_i} x_{n_i}$
θ_{n_1} θ_{n_2} $\cdots$ θ_{n_k}
$\hat{r}_n = \sum_{i=1}^{k} \phi_{n_i} x_{n_i}$
ϕ_{n_1} ϕ_{n_2} $\cdots$ ϕ_{n_k}
x_{n_1} x_{n_2} $\cdots$ x_{n_k}

Figure 3.1. *Block diagram of an SRV unit showing the various computations performed. The flow of signals used to compute the output of the unit is shown with solid lines; the signals used for learning are shown with dashed lines.*

belongs to a family of parametrized distributions, with parameters y and $\mathbf{x}$. Let

$$M(y,\mathbf{x}) = \int_{-\infty}^{\infty} r dH(r \mid y, \mathbf{x}) \tag{3.5}$$

be the regression function corresponding to this family of distributions (i.e., $M(y,\mathbf{x}) = E\{r \mid y, \mathbf{x}\}$). We assume that $M(.)$ is measurable and continuously differentiable almost everywhere.

The SRV unit uses the following algorithm to update the parameter vector $\boldsymbol{\theta}$, thus generating a sequence of random vectors $\boldsymbol{\theta}_n$:

$$\begin{aligned}
\boldsymbol{\theta}_{n+1} &= \boldsymbol{\theta}_n + \sigma_n \left(r(y_n, \mathbf{x}_n) - \hat{r}_n\right)(y_n - \mu_n)\mathbf{x}_n, && (3.6)\\
&\text{where } \mu_n = \boldsymbol{\theta}_n^{\mathsf{T}} \mathbf{x}_n, && (3.7)\\
&\sigma_n = s(\hat{r}_n), && (3.8)\\
&\hat{r}_n = \boldsymbol{\phi}_n^{\mathsf{T}} \mathbf{x}_n, \text{ and} && (3.9)\\
&y_n \sim N(\mu_n, \sigma_n). && (3.10)
\end{aligned}$$

The symbol $\sim$ is used to indicate that the quantity on the left hand side is a random variable with the distribution shown on the right hand side. Equation (3.6) can be rewritten as

$$\boldsymbol{\theta}_{n+1} = \begin{cases} \boldsymbol{\theta}_n + \sigma_n^2 g_n(\boldsymbol{\theta}_n, \boldsymbol{\phi}_n, \mathbf{x}_n)\mathbf{x}_n & \text{if } \sigma_n > 0, \\ \boldsymbol{\theta}_n & \text{if } \sigma_n = 0, \end{cases} \tag{3.11}$$

where

$$g_n(\boldsymbol{\theta}_n, \boldsymbol{\phi}_n, \mathbf{x}_n) = \left(r(y_n, \mathbf{x}_n) - \hat{r}_n\right)\left(\frac{y_n - \mu_n}{\sigma_n}\right), \tag{3.12}$$

and $\mu_n, \sigma_n, \hat{r}_n,$ and y_n are as defined in (3.7), (3.8), (3.9), and (3.10) respectively.[1]

An intuitive explanation for these update equations is the following. We can view the fraction in (3.12) as the *normalized perturbation* that has been added to the mean output of the unit for the given input. If this perturbation has caused the unit to receive an evaluation signal that is *more* than the expected evaluation, then it is

[1]We should note here that faster learning might be achieved in practice if the term σ_n^2 in (3.11) is replaced by a constant learning rate parameter $\alpha > 0$.

desirable for the unit to produce an output closer to the current output y_n. The mean output value should therefore move in the direction of the perturbation. That is, if the perturbation is positive, the unit should update its parameters so that the mean value increases. Conversely, if the perturbation is negative, the parameters should be updated so that the mean value decreases. On the other hand, if the evaluation received is *less* than the expected evaluation, then the unit should adjust its mean in the direction *opposite* to that of the perturbation. Equations (3.11) and (3.12) have this effect on the mean output.

Updating of the parameter vector ϕ used for computing the expected evaluation is relatively straightforward. For this, we want to associate with each input vector a corresponding reinforcement value, and because both the input vector and the reinforcement value are supplied to the unit, we can use the LMS rule of Widrow & Hoff (1960) to learn this association:

$$\phi_{n+1} = \phi_n + \rho\,(r(y_n, \mathbf{x}_n) - \hat{r}_n)\,\mathbf{x}_n, \tag{3.13}$$

where $\rho > 0$ is a learning rate parameter. This completes the description of the SRV algorithm in its original form.

Several algorithms related to the SRV algorithm have been described in the literature. Williams [152, 153] describes a very similar algorithm involving the use of the Gaussian distribution to generate real-valued outputs. His approach, unlike ours, is based on using the derivatives of the logarithm of the Gaussian distribution function to update the mean and standard deviation. Williams shows that, on average, using these derivatives to update the parameters results in an increase in the expected evaluation from the environment. Our algorithm also has components analogous to an earlier algorithm of Sutton [135], which also uses the Gaussian distribution to generate real-valued outputs. However, in his algorithm, the output is controlled by varying the mean alone; the standard deviation is held constant. Harth and Tzanakou [56], in a somewhat similar approach, designed an algorithm, called Alopex, to produce real values that are proportional to a bias with random noise added to it. In the original algorithm, the noise had a fixed distribution, although in more recent versions feedback is used to adjust the noise distribution. Farley and Clark [38] also described a scheme in which the noise level is varied based on changes in performance over previous time steps. More recently, Alspector, Allen, Hu, and Satyanarayana [5] described a reinforcement learning algorithm based on the Boltzmann machine learning algorithm [1], in which the noise in the activation of their stochastic units is controlled so as to keep the units active for a reasonable fraction of the time. In addition to the frequency of activation of the units, they also used measures such as the sum of the magnitudes of a unit's weights and the past history of reinforcement to determine the noise amplitude. Techniques involving perturbation of the weights of a unit rather than its output have also appeared recently in the literature [54, 62]. Although the net effect of perturbing the weights is to perturb the output of the unit, the former is considered easier to implement in hardware realizations of units.

3.4 Convergence properties of SRV units

In this section, we state and prove a convergence theorem that implies a form of optimal performance for a modified version of the SRV algorithm on associative reinforcement learning tasks. The proof of the theorem is based on Martingale theory and is modeled after Gladyshev's [44] proof of convergence of the Robbins-Monro process [113]. Modifications to the original SRV algorithm (Equations (3.6)–(3.13)) were necessary to make rigorous analysis possible. Nevertheless, we informally argue that the original algorithm has convergence properties similar to those of the modified algorithm, and present simulation results that support this claim.

It can be seen from the description of the SRV algorithm in Section 3.3 that it involves a complex interaction between the two concurrent learning processes, namely, the process for learning the expected evaluation for a given input and the process for learning the optimal output for that input. This interaction makes rigorous analysis of the learning system as a whole difficult. Because our primary interest here is in learning optimal outputs for given inputs, we make the following simplifications and assumptions to make the analysis tractable.

3.4.1 Simplifications

S1: In the SRV algorithm, the parameter vector ϕ_n defines the estimate, $\hat{r}_n$, of the expected evaluation given the mean output, μ_n, for an input, x_n. As discussed above, this estimation process interacts in a complex fashion with the process of estimating the optimal output, making analysis difficult. We therefore decided to simplify the algorithm by eliminating the internal estimation of the expected evaluation. Instead, at each iteration, the simplified SRV algorithm obtains this estimate directly from the environment. It does so by emitting another output, which is the mean μ_n, and receiving an evaluation $r(\mu_n, \mathrm{x}_n) \sim H(r \mid \mu_n, \mathrm{x}_n)$, which it uses instead of $\hat{r}_n$ in (3.6). The usual output y_n is also output to the environment, and the resulting evaluation $r(y_n, \mathrm{x}_n) \sim H(r \mid y_n, \mathrm{x}_n)$ is used as before in the update equation. Because it is no longer required for computing $\hat{r}_n$, we discarded the parameter vector ϕ.

S2: Because the simplified SRV unit no longer computes $\hat{r}_n$ internally, it does not compute the standard deviation σ_n as a function of $\hat{r}_n$. Instead, it uses a fixed sequence of real numbers $\{\sigma_n\}$, with the properties stated below.

Given these simplifications, the equations describing the generation of the random vector sequence $\{\theta_n\}$ are

$$\theta_{n+1} = \theta_n + \sigma_n \left(r(y_n, \mathrm{x}_n) - r(\mu_n, \mathrm{x}_n)\right)(y_n - \mu_n)\mathrm{x}_n, \tag{3.14}$$

$$\text{where } \mu_n = \theta_n^{\mathsf{T}}\mathrm{x}_n, \text{ and} \tag{3.15}$$

$$y_n \sim N(\mu_n, \sigma_n). \tag{3.16}$$

As before, we can define a function $g_n()$ so that (3.14) can be written as

$$\theta_{n+1} = \begin{cases} \theta_n + \sigma_n^2 g_n(\theta_n, \phi_n, \mathbf{x}_n)\mathbf{x}_n & \text{if } \sigma_n > 0, \\ \theta_n & \text{if } \sigma_n = 0, \end{cases} \tag{3.17}$$

where

$$g_n(\theta_n, \mathbf{x}_n) = (r(y_n, \mathbf{x}_n) - r(\mu_n, \mathbf{x}_n)) \left(\frac{y_n - \mu_n}{\sigma_n} \right), \tag{3.18}$$

and μ_n and y_n are as defined in equations (3.15) and (3.16) respectively.

3.4.2 Assumptions

The following assumptions are necessary to define the optimal output for each input in X, and to ensure that the random vector sequence θ_n converges to the corresponding optimal parameter values:

A1: The sequence of real numbers $\{\sigma_n\}$ is such that

$$\sigma_n \geq 0, \qquad \sum_{n=1}^{\infty} \sigma_n^3 = \infty, \qquad \sum_{n=1}^{\infty} \sigma_n^4 < \infty.$$

An example of such a sequence is $\left\{1/n^{1/3}\right\}$. Note that these three conditions imply that $\sigma_n^3 \to 0$. In the limit, therefore, the standard deviation of the output of the SRV unit goes to zero (see S2 above), and the output of the unit equals the mean μ_n used to generate it.

A2: 1. The input vectors $\mathbf{x}_n$ are drawn randomly from a finite set of k linearly independent vectors, $X = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(k)}\}$, and

2. $\Pr\{\mathbf{x}_n = \mathbf{x}^{(i)}\} = \xi^{(i)} > 0, \;\; 1 \leq i \leq k$.

Thus each vector in X has a non-zero probability of being chosen at any time.

A3: For each $\mathbf{x}^{(i)}$, $1 \leq i \leq k$, there is a unique real number $\beta^{(i)} \in Y$ such that

$$M(\beta^{(i)}, \mathbf{x}^{(i)}) \;=\; \max_{y} M(y, \mathbf{x}^{(i)}).$$

This assumption implies that the regression function of the environmental evaluation $M(.)$ has a unique maximum for each given input $\mathbf{x}^{(i)}$. It is also assumed that $M(y,\mathbf{x})$ has bounded first and second order derivatives with respect to y. Let

$$R(y,\mathbf{x}) = \frac{\partial M(y,\mathbf{x})}{\partial y}. \tag{3.19}$$

A4: For each $\mathbf{x}^{(i)}$, $1 \leq i \leq k$,

$$\sup_{\epsilon \leq |y-\beta^{(i)}| \leq \frac{1}{\epsilon}} (y-\beta^{(i)})\, R(y,\mathbf{x}^{(i)}) < 0, \tag{3.20}$$

for all $\epsilon > 0$. This assumption says that for each i, $R(y,\mathbf{x}^{(i)})$ behaves linearly as a function of y for values of y near $\beta^{(i)}$.

A5: Finally, we assume that

$$\int_{-\infty}^{\infty} \left(r(y,\mathbf{x}^{(i)}) - M(y,\mathbf{x}^{(i)})\right)^2 dH(r \mid y,\mathbf{x}^{(i)}) \leq h(1+(y-\beta^{(i)})^2) \tag{3.21}$$

for some real number $h > 0$. This assumption assures that the noise in the evaluative input $r(.)$ has a bounded variance and that $||r(y,\mathbf{x}^{(i)})||^2$ is bounded by a quadratic function for all y and $1 \leq i \leq k$.

Assumptions A1, A3, A4, and A5 are fairly standard in stochastic approximation literature [122, 34]. Of these, A3 and A5 can be easily met by most evaluation functions, while assumption A4 is more restrictive because it requires $R(.)$ to be linear in the neighborhood of the maximum for each input. It may be possible, however, to weaken this assumption as has been done in stochastic approximation literature, but only at the cost of complicating the analysis. For the sake of simplicity, we chose not to attempt to do so here.

Having delineated the above simplifications and assumptions, we are now ready to state the convergence theorem for the random process defined by (3.14).

Theorem 1 Given assumptions A1 - A5 and a finite random initial vector $\boldsymbol{\theta}_1$, the sequence of random vectors $\{\boldsymbol{\theta}_n\}$ generated by (3.14) converges with probability 1 to the unique vector $\boldsymbol{\theta}^* \in \mathrm{Ra}(A) + \boldsymbol{\theta}_1$ which satisfies the equations

$$\mu_i = \boldsymbol{\theta}^{*\mathsf{T}}\mathbf{x}^{(i)} = \beta^{(i)}, \quad 1 \leq i \leq k. \tag{3.22}$$

This theorem says that the *mean output* μ_i of the algorithm for any given input $\mathbf{x}^{(i)}$ converges to the optimal output $\beta^{(i)}$ with probability one. But since the standard deviation used for computing the output of the algorithm tends to zero, the output converges in probability to the mean output. Hence the above theorem implies that the output of the algorithm for any given input $\mathbf{x}^{(i)}$ converges in probability to the optimal output value $\beta^{(i)}$. The proof of the theorem is given in Appendix A.

3.4.3 Discussion

A few observations of general interest can be made regarding the above convergence theorem and the associated assumptions and simplifications. Most of these pertain to the relationship between the original SRV algorithm and the modified version to which the convergence result applies.

1. To make rigorous analysis possible, we made simplifications S1 and S2 to the original SRV algorithm. In doing so, we eliminated the need for estimating the expected evaluations $\hat{r}$ using the parameter vector $\boldsymbol{\phi}$. In the original algorithm, however, the two interacting processes—one for estimating the expected evaluation, and the other for estimating the optimal output—run simultaneously. Because the estimate of the expected evaluation has a critical role in the estimation of the optimal output, it is necessary to ensure the correctness of this estimate. Therefore one can expect stable convergence of the original SRV algorithm only

if the output estimation process is slower than the process for estimating the expected evaluation.

2. Assumption A1 places restrictions on the choice of the sequence $\{\sigma_n\}$ that are critical for the proof. Therefore this assumption cannot be discarded. But if we try to obtain an intuitive understanding of the role played by these restrictions in the convergence process, we can see that a similar effect is achieved by the method used to generate the σ_n sequence in the original SRV algorithm. The condition that $\sum_{n=1}^{\infty} \sigma_n^3 = \infty$ ensures that the sum of increments to the initial parameter vector $\boldsymbol{\theta}_1$ can be arbitrarily large, so that *any* finite $\boldsymbol{\theta}_1$ can be transformed into the optimal vector $\boldsymbol{\theta}^*$. At the same time, the condition that $\sum_{n=1}^{\infty} \sigma_n^4 < \infty$ ensures that the variance in $\boldsymbol{\theta}_n$ is finite and hence the vector cannot diverge to infinity. The σ_n sequence used in the original SRV algorithm also ensures the former consequence because σ_n does not become zero unless the optimal outputs are being produced for all inputs. Conditions on the computation of σ_n (Equations (3.8) and (3.9)) which will ensure boundedness of $\boldsymbol{\theta}_n$ when using the original SRV algorithm are currently unclear.

3. It is likely that linear independence of the input vectors, as required by A2, is not an essential condition for convergence. This is suggested by simulations of these algorithms in various tasks. We present some of these simulations in the next section. By defining an additional evaluation criterion for the parameter vector (for example, the least-squares criterion), it may be possible to show convergence to the best approximation of optimal performance relative to this criterion.

3.5 Some learning experiments using SRV units

In this section, we present simulations of the original SRV algorithm and the modified version for which the convergence result above applies in two simple associative reinforcement learning tasks. The purpose of these simulations is twofold: first, to illustrate the convergence properties of the algorithms, and second, to examine the practical significance of the simplifications and assumptions made in Section 3.4.1. To do this, we have designed the tasks so that all the conditions of the theorem are met in the first task, while some of these are violated in the second task. Clearly, associative reinforcement learning tasks can be made very difficult to solve, for example, by increasing the number of input vectors or their dimensionality. We have deliberately chosen rather simple tasks here so that the important aspects of the algorithm's performance are readily discernible. However, several examples of networks with SRV units that can solve fairly complex tasks are presented in the succeeding chapters.

Each task is defined by a finite set of input vectors, X, and their corresponding β values (see assumption A3). For Task 1, X contains three vectors, $\mathbf{x}^{(1)}$, $\mathbf{x}^{(2)}$, and $\mathbf{x}^{(3)}$, which are linearly independent but not orthogonal. The set of input vectors defining Task 2 has the same three vectors as Task 1 plus two more. These five vectors together constitute a linearly dependent set (specifically, $\mathbf{x}^{(5)} = 0.5(\mathbf{x}^{(1)} + \mathbf{x}^{(3)})$.) The vectors in the input set for a task are all equally likely to occur at each time step. Thus $\xi^{(i)} = 1/3, \quad i = 1,2,3$ for Task 1, and $\xi^{(i)} = 1/5, \quad i = 1,\cdots,5$ for Task 2. These conditions ensure that A2 is satisfied by the set of vectors for Task 1. For each run of the simulation, we picked the initial parameter vector $\boldsymbol{\theta}_1$ (and $\boldsymbol{\phi}_1$ for the SRV algorithm) randomly from a uniform distribution over $[-0.5, 0.5]^4$.

The sequence $\{\sigma_n\}_{n=1}^{\infty}$ used by the modified SRV algorithm was defined in our simulations as

$$\sigma_n = \frac{1}{(\lfloor n/20 \rfloor)^{1/3}}, \tag{3.23}$$

where $\lfloor\ \rfloor$ denotes the *floor* function. This sequence satisfies the conditions of A1, as the reader can easily verify. In the case of the SRV algorithm, we first computed $\hat{r}$ as

$$\hat{r}_n = f(\boldsymbol{\phi}_n^{\mathrm{T}} \mathbf{x}_n), \tag{3.24}$$

where

$$f(a) = \frac{1}{1 + e^{-a}} \tag{3.25}$$

is the logistic function that maps the real line onto the interval $(0, 1)$. Using this, we computed σ_n for the SRV algorithm simply as

$$\sigma_n = s(\hat{r}_n) = 1 - \hat{r}_n. \tag{3.26}$$

In these simulations, we also set the learning rate, ρ_n, for the parameter vector ϕ (see (3.13)) to be constant and equal to 0.5.

Two different evaluation functions, r_1 and r_2, were used for the simulations. These are defined as follows:

$$r_1(y, \mathbf{x}^{(i)}) = 1.0 - \mid f(\beta^{(i)}) - f(y) \mid, \tag{3.27}$$

and

$$r_2(y, \mathbf{x}^{(i)}) \sim N(r_1(y, \mathbf{x}^{(i)}), 0.1), \tag{3.28}$$

where $f(.)$ is the logistic function defined above, and $N(\mu, \sigma)$ is the Gaussian distribution function. Note that r_1 is a deterministic evaluation function, while r_2 returns a random evaluation whose mean is the evaluation returned by r_1. The standard deviation used in computing r_2 is 10% of the range of r_1, which is a number between 0 and 1. Therefore r_2 is a very noisy version of r_1. From these definitions, it is clear that the highest *expected* reinforcement, given an input vector, is 1.0 for either evaluation function. It is also easy to verify that both these evaluation functions satisfy assumptions A3, A4, and A5.

Performance of the two algorithms was measured by recording a smoothed reinforcement value at each time step over the course of a training run. The smoothed reinforcement at any time step was computed as the average of the reinforcement received over the last 100 consecutive time steps. This measure conveys more information about the change in performance of the algorithms over time than the actual reinforcement received at a time step. Perhaps a more accurate performance measure would be the expected value of the reinforcement at each time step, given the parameters and the input vector at that time step. However, because both the output and the reinforcement are continuous-valued random numbers with their own distributions, obtaining a closed form expression for this expected value is not easy. The moving average of past reinforcements is a good approximation of this expected value, and one that is easily computable. A single simulation run lasted for 4500 time steps for Task 1 and 7500 time steps for Task 2. If each input vector in X were presented sequentially to the learning algorithm, these simulation durations would correspond to 1500 presentations of the entire input set for either task. For the purposes of collecting statistics, we conducted 25 runs of each algorithm in each task with each evaluation function.

A. Task 1

For Task 1, the input vectors are $\mathbf{x}^{(1)} = (1,1,1,0)^{\mathsf{T}}$, $\mathbf{x}^{(2)} = (1,1,1,1)^{\mathsf{T}}$, and $\mathbf{x}^{(3)} = (1,0,1,1)^{\mathsf{T}}$, and the corresponding vector of optimal output values is $\boldsymbol{\beta} = (-1.3862, 1.3862, 0.0)^{\mathsf{T}}$. The learning system receives the maximum reinforcement when it responds to an input vector with the corresponding optimal output. Note that because these input vectors are linearly independent and the initial parameter vector $\boldsymbol{\theta}_1$ is finite, it follows from Lemma 1 that there exists an optimal parameter vector $\boldsymbol{\theta}^*$ that is unique for that initial parameter vector.

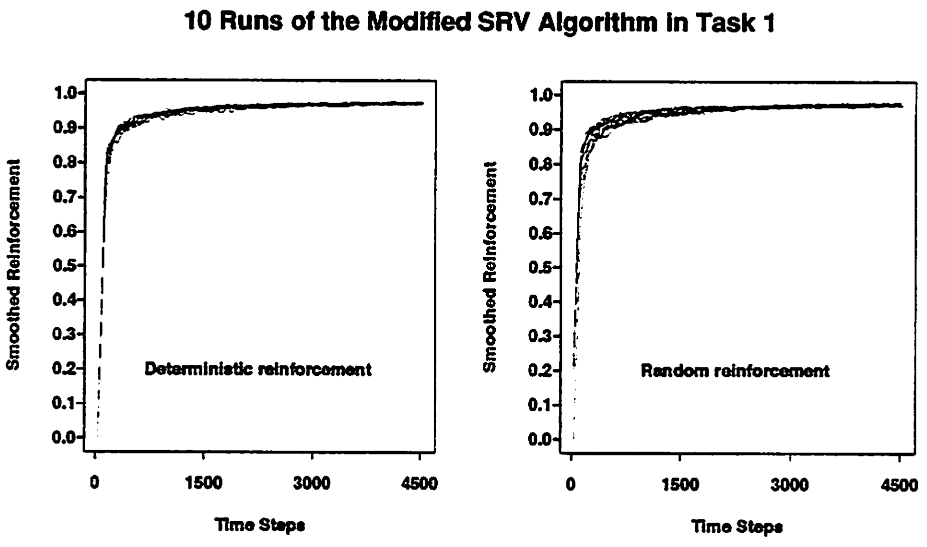
The results of simulating the modified SRV algorithm in Task 1 with both evaluation functions are shown in Figure 3.2. Graphs in the figure are plots of the smoothed reinforcement at each time step n, $1 \leq n \leq 4500$. Figure 3.2a shows results of 10 individual runs with each type of reinforcement. The average smoothed reinforcement over 25 runs for each type of reinforcement is plotted in Figure 3.2b. Comparing these plots, it is clear that the algorithm performs almost as well with random reinforcement as with deterministic reinforcement. Further, the individual runs all show identical convergence behavior. Since all the assumptions A1–A5 made for the theorem are satisfied by the definition of Task 1, these plots serve to illustrate the convergence behavior assured by the theorem. For example, in one of the simulation runs with random reinforcement, the randomly chosen value for $\boldsymbol{\theta}_1$ was $(0.3163, 0.0153, -0.3275, -0.3472)^{\mathrm{T}}$. After 4500 time steps, $\boldsymbol{\theta}_{4500}$ equaled $(-1.0263, 1.3097, -1.6702, 2.6557)^{\mathrm{T}}$, which is close to the theoretical asymptotic value[2] $\boldsymbol{\theta}^* = (-1.0644, 1.3863, -1.7082, 2.7726)^{\mathrm{T}}$ *for the above value of* $\boldsymbol{\theta}_1$. The smoothed reinforcement at this time step was 0.9697, which is also close to the optimal value of 1.0. Obviously, more training steps would be required for closer convergence. Moreover, because σ_n tends to zero very rapidly, longer and longer training sequences become necessary for comparable reductions in the deviation from the optimal parameters.

An identical set of plots of simulations of the original SRV algorithm in Task 1 are presented in Figure 3.3. Recall that in this version of the algorithm, an additional set of parameters is used to learn the expected reinforcement, which is then used to compute the standard deviation σ_n of the output. Because of the uncertainty involved in this process, the SRV algorithm is slower than the modified SRV algorithm. Nevertheless, it exhibits similar convergence of the parameters to their optimal values. Further, Figure 3.3b shows that the SRV algorithm also performs equally well with both deterministic and noisy reinforcements. Thus, the simplifications S1 and S2 introduced to facilitate theoretical analysis do not appear essential for convergence in practice, at least in the simple tasks presented here. Moreover, the simplifications do not seem to significantly improve the algorithm's performance in these tasks.

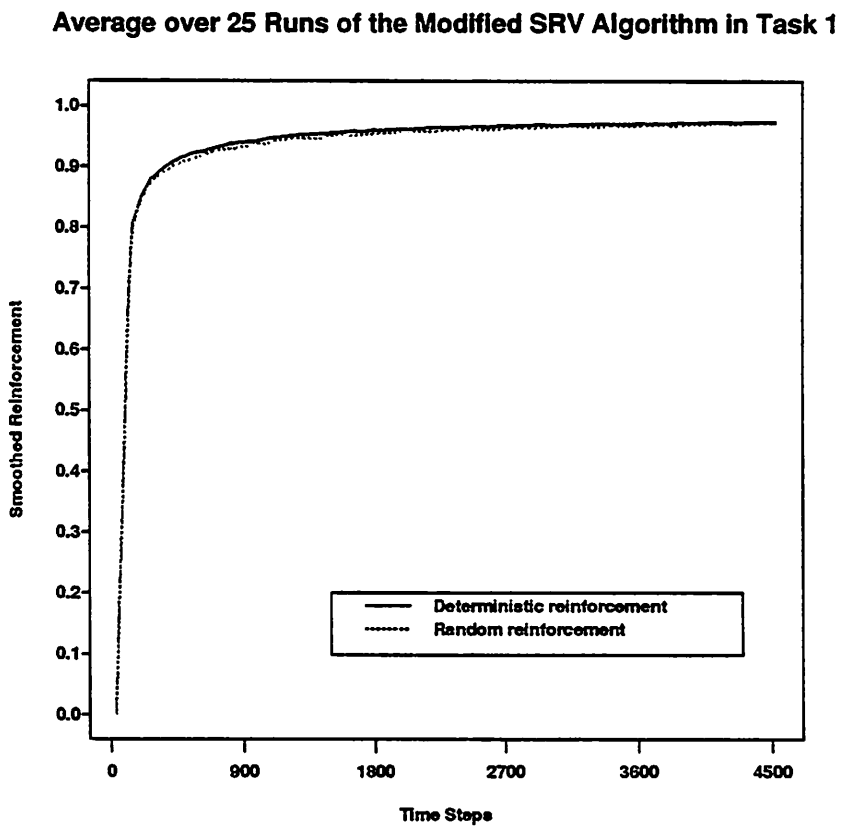
B. Task 2

For Task 2, the input set for Task 1 was augmented with two additional vectors. These are $\mathbf{x}^{(4)} = (1, 2, 1, 1)^{\mathrm{T}}$ and $\mathbf{x}^{(5)} = (1, 0.5, 1, 0.5)^{\mathrm{T}}$. The corresponding vector of optimal output values is $\beta = (2.7726, -0.6931)^{\mathrm{T}}$. As noted above, with the additional input vectors, X is no longer a linearly independent set and hence assumption A2 is violated. Therefore it is no longer possible to use the procedure

[2]The proof of Lemma 1 shows how this theoretical asymptotic value can be computed.

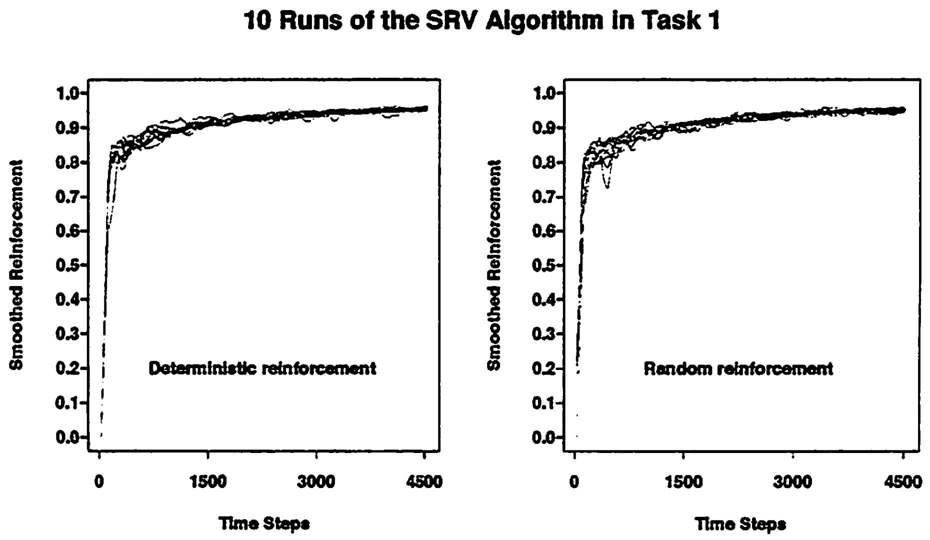


(a)

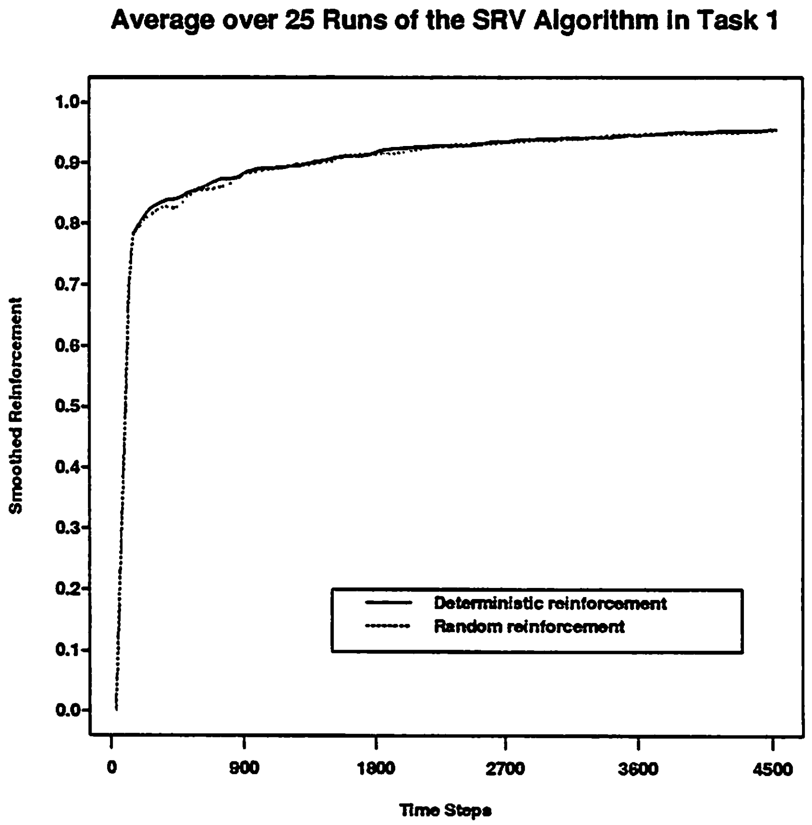


(b)

Figure 3.2. Simulations results for the modified SRV algorithm in Task 1. (a) Graphs of the smoothed reinforcement for ten runs with each type of reinforcement. (b) Curves showing the smoothed reinforcement averaged over 25 runs with each type of reinforcement.



(a)



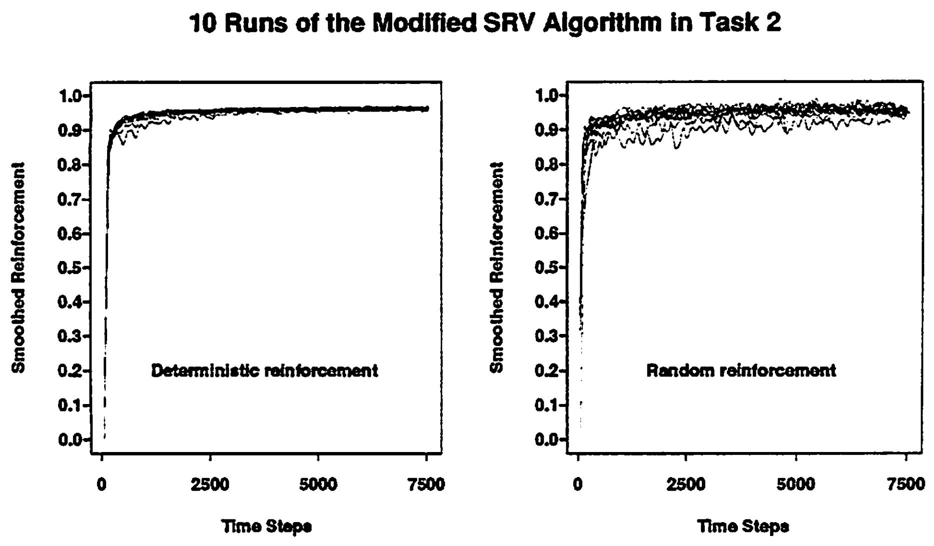
(b)

Figure 3.3. Simulations results for the SRV algorithm in Task 1. (a) Graphs of the smoothed reinforcement for ten runs with each type of reinforcement. (b) Curves showing the smoothed reinforcement averaged over 25 runs with each type of reinforcement.

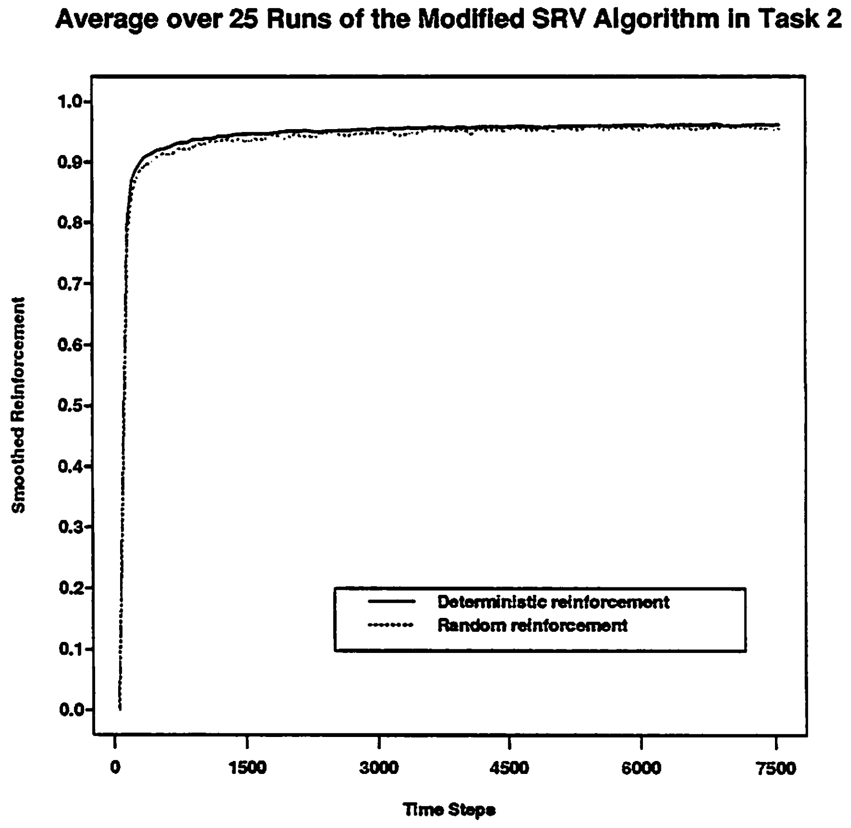
in Lemma 1 to determine the optimal parameter vector. To ensure that there is at least one set of parameters which maximizes the expected reinforcement, we chose the optimal output values for the new vectors in the following manner. Using the original three input vectors and their corresponding optimal output values, we computed the optimal parameter vector $\boldsymbol{\theta}^*$ assuming that $\boldsymbol{\theta}_1 = \mathbf{0}$. This yields $\boldsymbol{\theta}^* = (-1.3863, 1.3863, -1.3863, 2.7726)^{\mathsf{T}}$. We then set $\beta^{(i)} = \boldsymbol{\theta}^{*\mathsf{T}}\mathbf{x}^{(i)}$ for $i = 4, 5$. Clearly, $\boldsymbol{\theta}^*$ is the only parameter vector that can yield optimal outputs for all five inputs, and hence it is the optimal value for the parameters regardless of the starting value $\boldsymbol{\theta}_1$.

Figures 3.4 and 3.5 show the results of simulating the modified SRV algorithm and SRV algorithm respectively in Task 2. As before, part (a) of each figure depicts the performance over 10 runs with each type of reinforcement, while part (b) depicts the performance averaged over 25 runs. Comparing these figures with those for Task 1, we can see that the individual runs are more jittery, especially in the case of random reinforcement. However, all runs of the simulation converged in the case of both algorithms, as is illustrated by the plots of performance averaged over 25 runs. One run using the SRV algorithm did not start to converge until after the 2500^{th} time step. The apparent reason for this was an unfortunate choice of the random initial parameters for $\boldsymbol{\phi}$ that led the algorithm to predict higher reinforcement values than were being actually obtained. Until the reinforcement predictor became more accurate, the algorithm could not learn the right outputs and converge. Observation of individual runs leads us to believe that the non-uniform convergence behavior is probably due to the random initial parameters, since, for this task, there is a single set of optimal parameters independent of the initial values. This also accounts for the slower overall convergence rate for Task 2 as compared with Task 1.

Figure 3.4 shows that the condition that the input vectors are linearly independent (assumption A2) is not critical for convergence of the modified SRV algorithm, as long as the task itself is linear. Using random reinforcement also appears to affect the rate

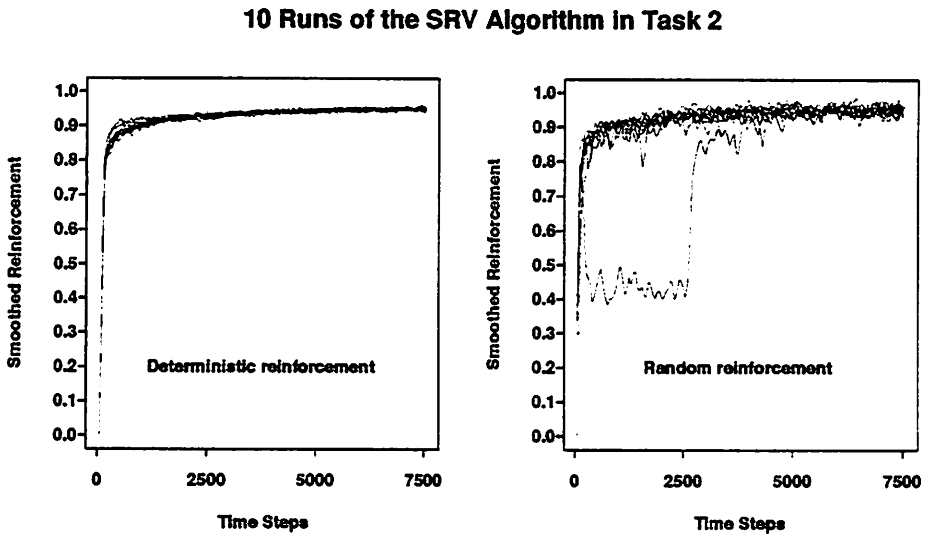


(a)

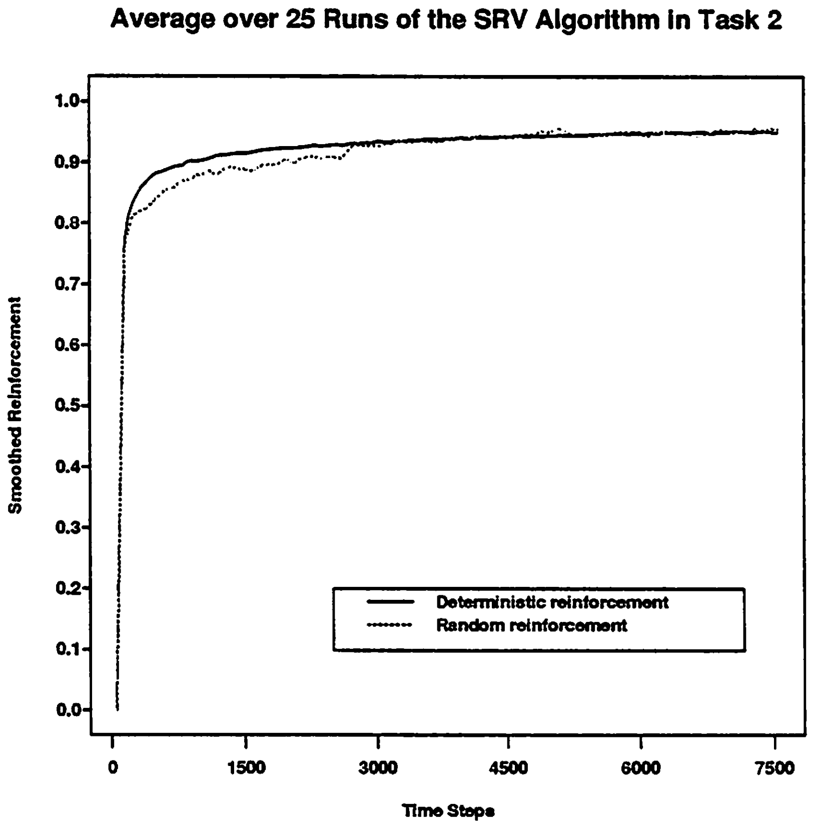


(b)

Figure 3.4. *Simulations results for the modified SRV algorithm in Task 2. (a) Graphs of the smoothed reinforcement for ten runs with each type of reinforcement. (b) The smoothed reinforcement averaged over 25 runs with each type of reinforcement.*



(a)



(b)

Figure 3.5. *Simulations results for the SRV algorithm in Task 2. (a) Graphs of the smoothed reinforcement for ten runs with each type of reinforcement. (b) The smoothed reinforcement averaged over 25 runs with each type of reinforcement.*

of convergence only marginally. The same is also true for the SRV algorithm. Although these results only cover the case of a finite number of input vectors, elsewhere [48, 50] we have presented simulations in which the SRV algorithm performs well even when the input vectors were drawn from an infinite set. These simulations show that the SRV algorithm appears to have all the desirable convergence properties proved for the modified SRV algorithm.

3.6 Summary

In this chapter, we have presented an algorithm that can learn associative maps from input vectors to real-valued actions without the necessity of having the desired responses available to the algorithm. This work extends the work of Barto and Anandan [18] in synthesizing associative reinforcement learning algorithms using techniques from pattern classification and automata theory. We have also presented a theorem implying convergence with probability one for a simplified version of the algorithm operating under certain conditions. Simulation results suggest that neither the simplifications nor some of the conditions of the theorem are essential for good convergence in practice. We feel that the central idea of the SRV algorithm, namely, using predictions of the outcomes of actions to incrementally optimize the actions, is an important one that merits further study. This idea was suggested before by Barto, Sutton, and Brower [24] and studied by Sutton [136]. Finally, the ability to learn real-valued functions using less informative training signals than conventional learning algorithms can to be a useful tool in several applications, as we shall demonstrate in the succeeding chapters.

CHAPTER 4

LEARNING CONTROL USING DIRECT REINFORCEMENT LEARNING

In this chapter, three control tasks are used as examples to illustrate the utility of direct associative reinforcement learning methods for learning control. The first of these, a peg-in-hole insertion task, is an example of learning fine-motion control under real-world conditions of uncertainty and noise. The second task is one of learning the inverse kinematics of a simulated under-constrained robot arm. The last task involves learning to control an unstable dynamic system, namely a cart-pole system. For all three tasks, the associative reinforcement learning method used was a connectionist network that produced control actions using an output layer of SRV units. In presenting the results of training the controllers on these tasks, an attempt has been made to compare and contrast the performance of the direct learning methods with that of indirect learning methods applied to the same task. Appendix B contains the implementational details of the controlled processes and the controllers for these three tasks.

4.1 Learning fine-motion control: A peg-in-hole insertion task

As a prototypical example of a task involving interaction between a robot and its environment, the peg-in-hole insertion task has been used by roboticists for testing various approaches to robot control. As a canonical robot assembly operation, the peg-in-hole insertion task is also very important in industrial robotics. Comprising about 33% of all automated assembly operations, peg-in-hole insertions are the most frequent assembly operations [105]. Both two-dimensional [146] and three-dimensional [51] peg-in-hole tasks have been studied in detail, and a number of approaches to the general problem have been proposed (see [46] for an overview). While the abstract peg-in-hole task can be solved quite easily, real-world conditions of uncertainty and noise can substantially degrade the performance of traditional control methods.

Sources of uncertainty and noise include (1) errors and noise in sensations, (2) errors in execution of motion commands, and (3) uncertainty due to movement of the part grasped by the robot. The first of these is the most common source of uncertainty. Position sensors can be especially unreliable when the robot is interacting with external objects, resulting, for example, in inaccurate knowledge of relative locations of parts of the robot with respect to the external objects. Noisy force sensations further compound the problem of control when the robot is interacting with external objects. Moreover, hybrid position/force controllers often exhibit instabilities near the boundary between contact and non-contact. The second source of uncertainty is due to the dynamics of the robot itself. Because of friction, gear backlash, etc., giving the same command in the same starting configuration can result in different motions of the robot. The third source of uncertainty stems from the possibility of parts moving within the grasp of the gripper. Most grippers have no means of detecting such motion.

Under such conditions, traditional methods do not perform very well, and the peg-in-hole problem becomes a good candidate for adaptive approaches [126]. Both direct and indirect learning methods are applicable. But given these conditions, hand-crafting or learning an adequate robot model—imperative if one is to use indirect methods for training the controller—can be very difficult. We argue that it is easier to use direct methods to learn a reactive control strategy that enables good performance under uncertainty and noise. This is demonstrated here by using a Zebra Zero robot shown in Figure 4.1 to perform peg-in-hole insertions. This robot is equipped with a wrist force sensor, and can also sense positions of the joints of the arm through position encoders.

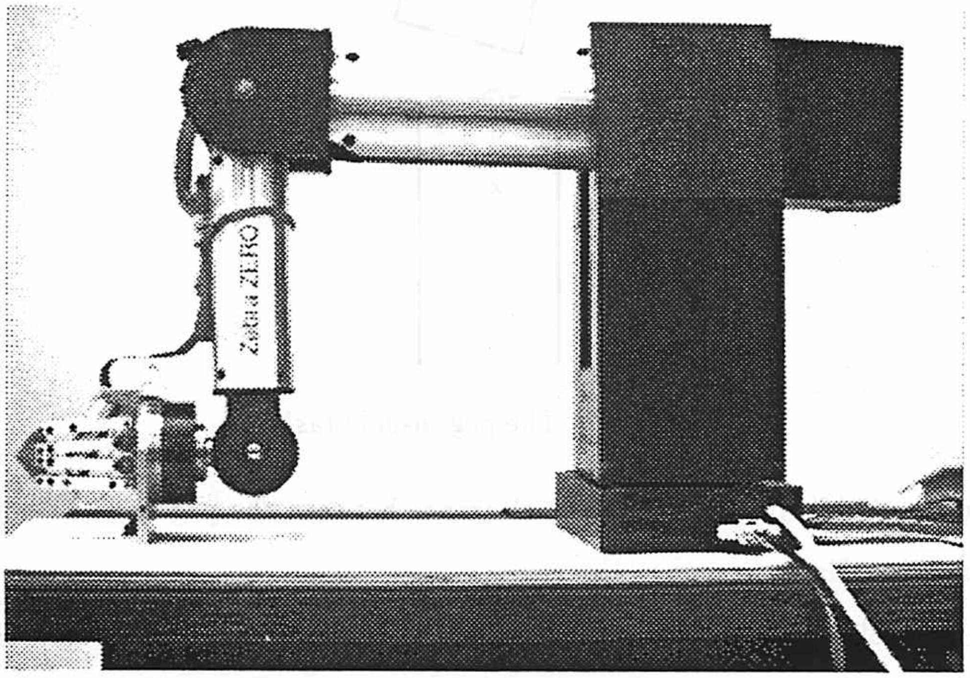


Figure 4.1. *The Zebra Zero robot used for the peg-in-hole insertion task.*

As an initial implementation, a two-dimensional version of the peg-in-hole task, depicted in Figure 4.2, was attempted. In this version of the task, six inputs are provided to the controller at each time step. These are the position of the peg (X, Y, Θ), computed from the sensed joint positions of the robot arm, and the force and moment sensations (F_x, F_y, M_z). Using these inputs, the controller has to compute a velocity command for that time step. Because it is acting in a closed loop with the robot, it is possible for the controller to learn a reactive, or closed-loop, control strategy for performing the insertion task. Figure 4.3 shows the network that was trained using direct reinforcement learning to perform peg insertions. In the experiments reported here, the peg used was 50mm long and 22.225mm (7/8in) wide, while the hole was

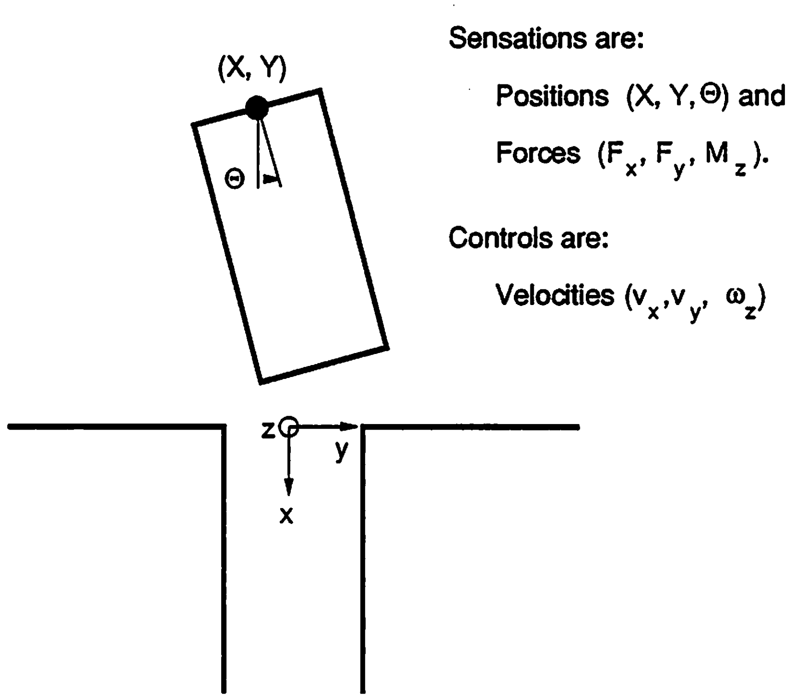

(X, Y)
Θ
Sensations are:
Positions (X, Y, Θ) and
Forces (F_x, F_y, M_z).
Controls are:
Velocities (v_x, v_y, ω_z)
z
y
x

Figure 4.2. *The peg-in-hole task.*

23.8125mm (15/16in) wide. Thus the clearance between the peg and the hole was 0.79375mm (1/32in).

The controller was trained in a sequence of training runs. Each training run started with the peg at a random position and orientation with respect to the hole and ended either when the peg was successfully inserted to a depth of 35mm into the hole, or when 100 time steps had elapsed. The following interaction took place between the controller and the robot at each time step during training. To begin, the sensed peg position and forces were input to the controller network, which used them to compute a control action. This action was executed by the robot, resulting in some motion of the peg. The network's output was then evaluated based on the new peg position and the forces acting on the peg. Two criteria were used in computing this evaluation, which ranged from 0 to 1, with 1 denoting the best evaluation. A base evaluation was computed first, using the discrepancy between the current sensed position of the peg and the desired position with the peg inserted in the hole. A penalty term was then subtracted from this evaluation when any of the sensed forces

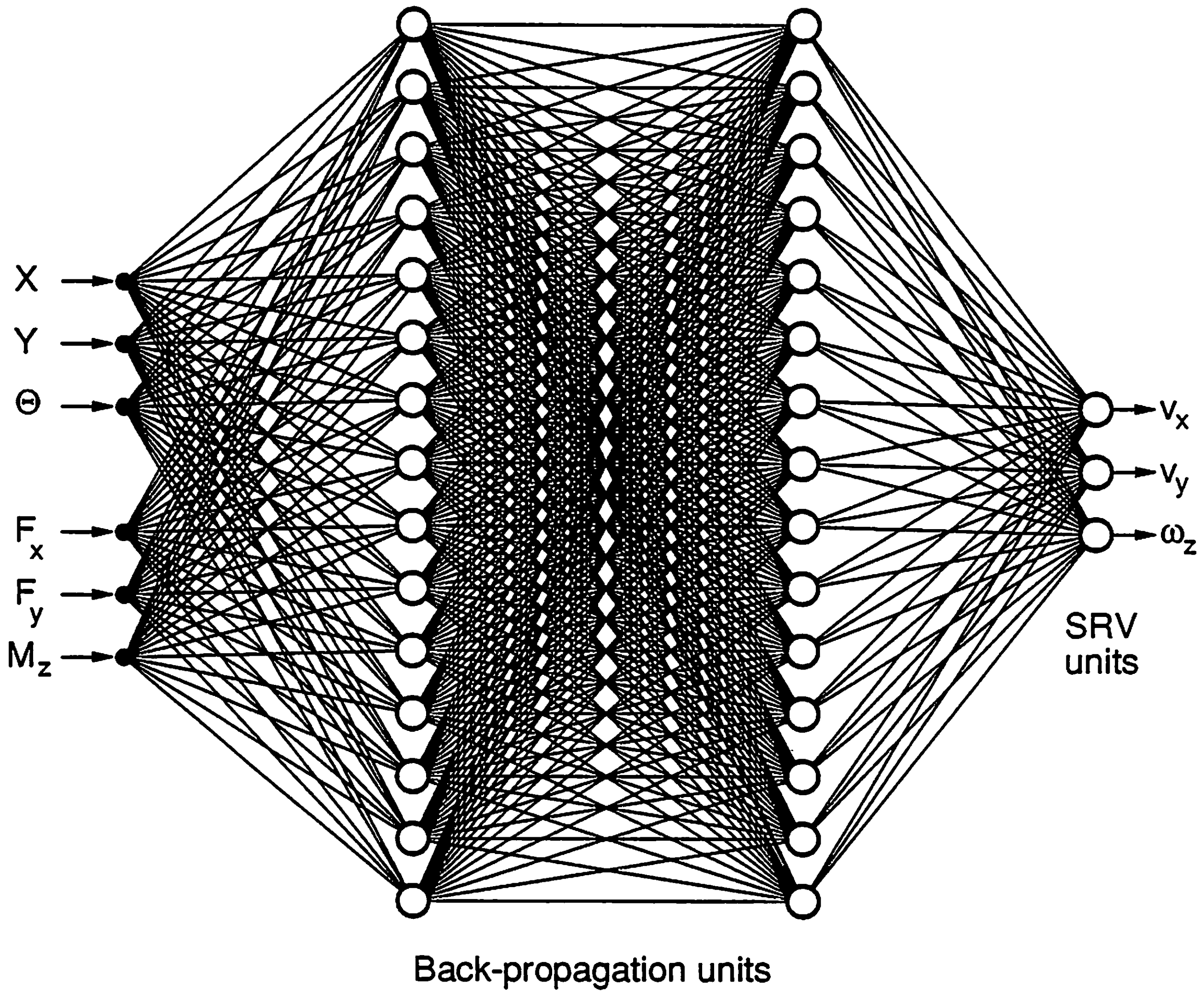


Figure 4.3. *The network used for the peg-in-hole task. The network has an input layer of 6 units, two hidden layers of back-propagation units and an output layer of 3 SRV units. All the units in a layer are connected to all the units in the succeeding layer.*

on the peg exceeded a preset maximum of 0.5 Newtons, thus lowering the evaluation. Using this evaluation, the controller network updated its weights and the cycle was repeated.

Results

A learning curve showing the final evaluation over 500 consecutive training runs is shown in Figure 4.4. The final evaluation levels off after about 150 training runs

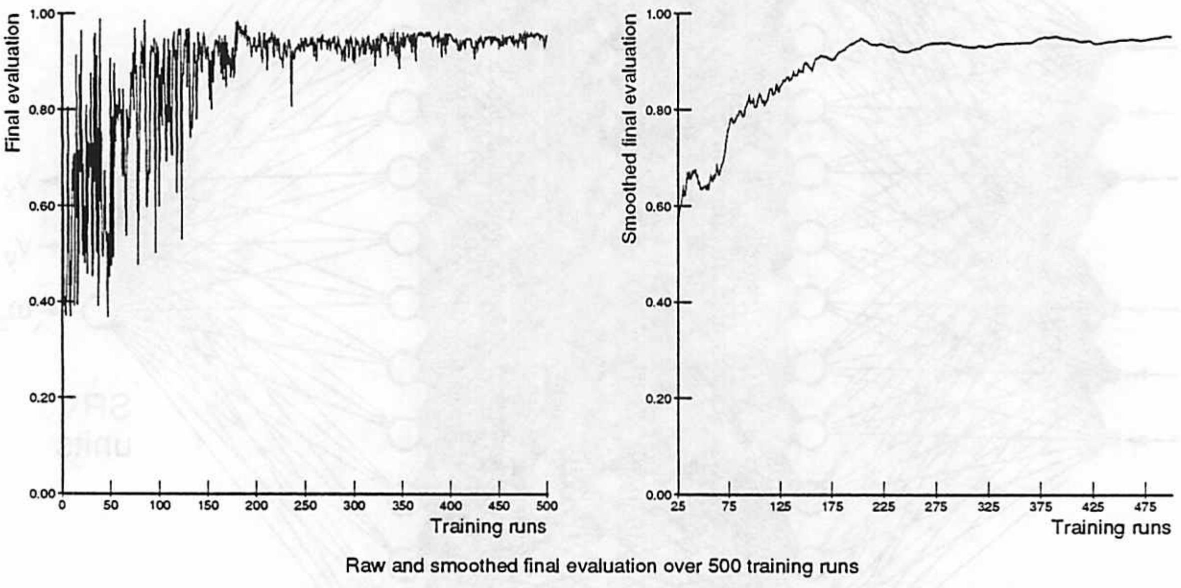


Figure 4.4. *Peg-in-hole task: Final evaluation received over 500 consecutive training runs. The smoothed curve was obtained by filtering the raw data using a moving-average window of 25 consecutive values.*

because after that amount of training, the controller is consistently able to perform successful insertions within 100 time steps. However, performance as measured by insertion time continues to improve, as is indicated by the learning curve in Figure 4.5, which shows the time to insertion decreasing continuously over the 500 training runs. These curves indicate that the controller becomes progressively more skillful at peg insertion with training. The performance of the controller was tested on 9 test cases after 100, 200, 300, 400, and 500 training runs. In the 9 test cases, insertion was

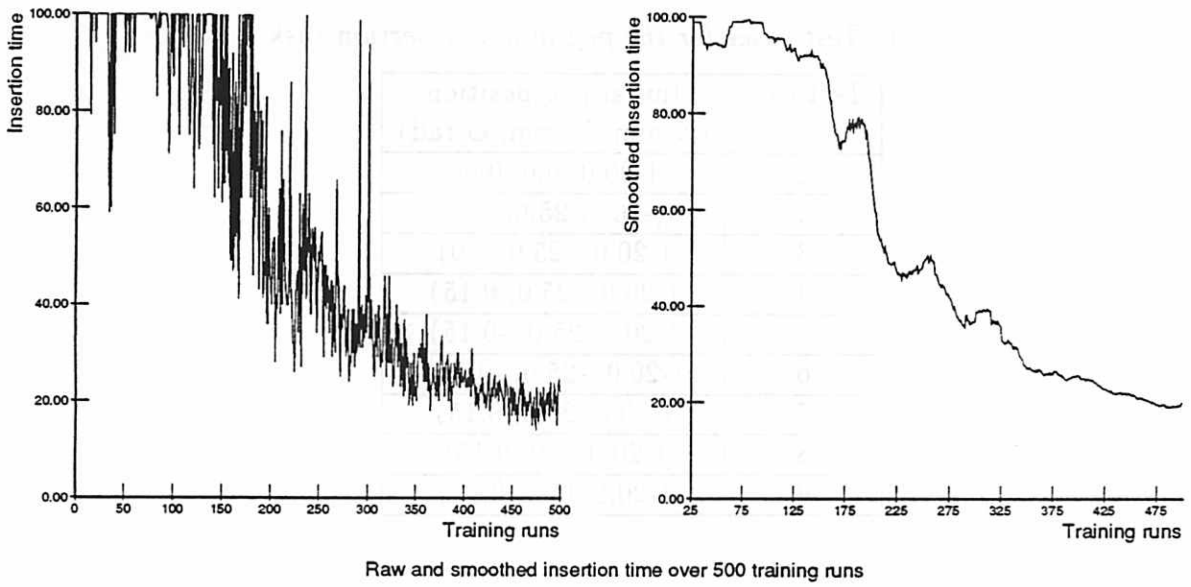


Figure 4.5. *Peg-in-hole task: Insertion time (in simulation time steps) taken on each of 500 consecutive training runs. The smoothed curve was obtained by filtering the raw data using a moving-average window of 25 consecutive values.*

attempted with the peg at one of 9 fixed initial locations and orientations, as listed in Table 4.1. An example of the performance (test case 6) is shown in Figure 4.6.

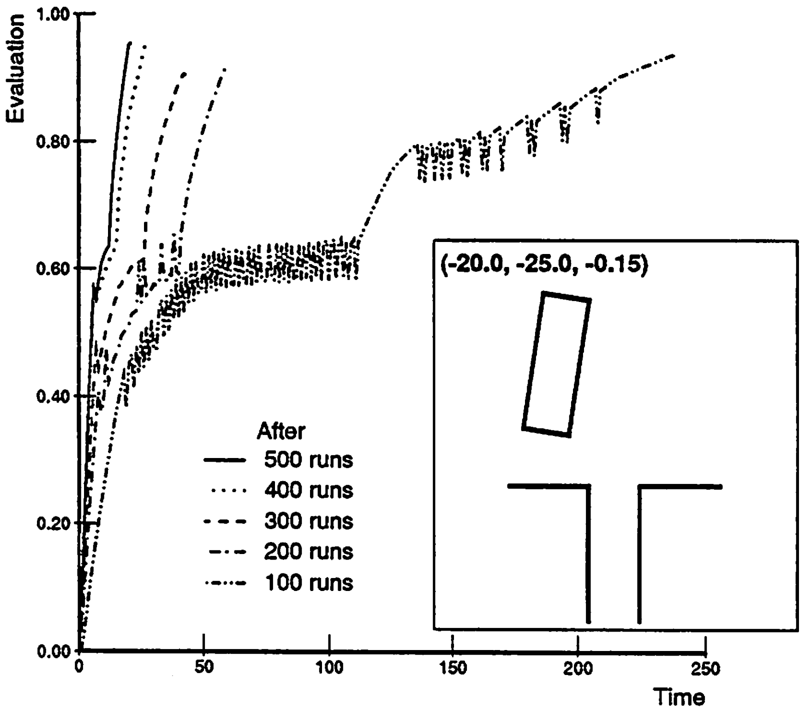
Discussion

The two-dimensional peg-in-hole task is one of the most widely studied robotics tasks. Consequently, the literature on the subject is vast and we cannot hope to do full justice to it in this discussion. Instead, we direct the interested reader to the references cited here and at the beginning of Section 4.1. Attention is restricted in this discussion to the proposed approaches to peg-in-hole insertion under conditions of uncertainty and noise. These approaches can be grouped into two major classes: methods based on off-line planning, and methods based on reactive control.[1]

[1]Although these methods have been developed for general robotic tasks involving physical interactions between the robot and its environment, for convenience, we discuss them from the point of view of peg-in-hole insertion.

Table 4.1. *Test cases for the peg-in-hole insertion task.*

Test case	Initial peg position (X mm, Y mm, Θ rad)
1	(-20.0, 0.0, 0.0)
2	(-20.0, 25.0, 0.0)
3	(-20.0, -25.0, 0.0)
4	(-20.0, -25.0, 0.15)
5	(-20.0, 25.0, -0.15)
6	(-20.0, -25.0, -0.15)
7	(-20.0, 25.0, 0.15)
8	(-20.0, 0.0, 0.15)
9	(-20.0, 0.0, -0.15)



Performance on the peg insertion task

Figure 4.6. *Peg-in-hole task: Performance on test case 6 after various intervals of training. The inset shows the initial position and orientation of the peg for this test case.*

Off-line planning methods combine geometric analysis of the peg-hole configuration with analysis of the task statics to determine motion strategies that will result in successful insertion [146, 51, 46]. In the presence of uncertainty in sensing and control, researchers have suggested incorporating the uncertainty into the geometric model of the task in configuration space. *Pre-images* [85] or *backprojections* [35] of the goal, i.e., sets of peg positions and commanded motions that would result in successful insertion, are then computed using the geometric model. If the peg is located within a pre-image, a motion command exists which will result in successful insertion. Frequently, however, the peg does not lie in any pre-image of the hole. In such an event, a recursive procedure can be used [85], wherein pre-images, starting with that of the task goal (i.e., the hole), are successively treated as goals for which new pre-images are computed. If a pre-image containing the peg is found at any stage in the recursion, then a trajectory can be planned through the successive pre-images that will result in successful insertion. Other strategies for off-line planning have also been proposed (e.g., [27, 30]).

Off-line planning is based on the assumption that a realistic characterization of the margins of uncertainty is available. While planning, only plans that assure successful insertion even at the limits of the margins of uncertainty are considered. Although it is possible to inflate the margins of uncertainty, thereby ensuring successful physical execution of the generated plan, allowing excessive margins can actually hamper planning by eliminating physically realizable solutions. Therefore the success of off-line planning seems to depend on accurate knowledge of the uncertainty, which is something of an oxymoron.

Methods based on reactive control, on the other hand, try to counter the effects of uncertainty with on-line modification of the motion control based on sensory feedback. *Compliant* motion control, in which the motion trajectory is modified by contact forces or tactile stimuli occurring during the motion, is often used, with the compliant behavior either being actively generated or occurring passively due to the physical characteristics of the robot. For the peg-in-hole task, compliance-based strategies are usually keyed on the forces due to interactions between the peg and the hole. The remote center compliance (RCC) device [146], for example, is a passive mechanical device that embodies a reactive control strategy wherein compliant motion due to forces on the peg is used to correct small initial positioning errors. However, humans find it quite difficult to specify compliant motions for a given task [85], and hence techniques for learning compliant behavior are very useful.

In an early attempt to learn a compliant control strategy, Simons et al. [126] used stochastic automata to learn corrections to the motion of the peg in a plane perpendicular to the direction of insertion, based on the forces sensed in that plane. The space of forces was divided up into a grid and, in each grid cell, one of four possible directions of corrective motion was learned using a stochastic automaton. In their experiments, Simons et al. used a specially designed wrist that sensed forces and had a programmable compliance. Although their approach was tested using an actual robot, Simons et al. [126] did not provide performance results or implementation details necessary for comparing their approach with the one presented in Section 4.1. More recently, Asada [9] used a multi-layer backpropagation network to learn a non-linear compliance mapping from forces to motion corrections. Asada trained the network in a supervised fashion using ten training pairs of force signatures and the associated corrective motions. However, no performance results were given, and it is not clear how well the learned compliance behavior improves peg-in-hole insertion in the presence of uncertainty.

In another recent study, Lee and Kim [82] used an expert system based on a paradigm called EARSA (Expert Assisted Robot Skill Acquisition) to learn skilled peg-in-hole insertions starting with an initial expert-specified rule base. As in our approach to the peg-in-hole task, uncertainty and noise were handled by learning a compliant control behavior based on position and force feedback. Although implemented in a symbolic system, their approach—in which random search among the expert's rules is used to discover the best control action in the face of uncertainty and noise—can be considered a direct method similar to ours. We compare these two approaches in some detail below.

Lee and Kim [82] reported experiments using a *simulated two-dimensional* peg and hole that demonstrated how their system could learn new rules that improved insertion speed and avoided jamming. Only three different initial peg locations were used. In their experiments, the peg used was 30.48mm long and 12.7mm (0.5in) wide, while the hole was 12.954 (0.51in) wide. Thus the clearance between the peg and the hole was 0.127mm (.005in). They also corrupted the simulated sensory feedback of positions and forces with mean-zero gaussian noise which had a standard deviation of 0.00127mm and was bounded to be less than 0.0127mm in magnitude. Note that the maximum position error is therefore an order of magnitude smaller than the clearance between the peg and the hole.

While the performance results reported by Lee and Kim [82] are comparable to those obtained using our approach, several factors suggest that our approach might have greater advantages. To begin with, Lee and Kim had to use discretized representations of the positions, forces, and velocities in their symbolic system, whereas we could use continuous-valued variables to represent these quantities in our learning system. This is a major advantage because one does not have to deal with the problem of selecting appropriate discrete representations for quantities that are naturally continuous. Second, Lee and Kim's controller started with an initial expert rule base that enabled it to perform insertions even before any learning took place. It is not clear how good this initial knowledge has to be for their approach to work in general. In contrast, our controller could learn skilled peg insertion starting with no initial control knowledge.

Our results also indicate that our approach works well with a physical system, despite the much higher magnitudes of noise and consequently greater degree of uncertainty inherent in dealing with physical systems. For example, while the maximum error in the sensed position of the peg was a degree of magnitude smaller than the clearance between the peg and the hole in Lee and Kim's simulated system, for the Zebra Zero, the position error was frequently more than three times the clearance. The noise in the force sensations returned by the Zebra Zero was also very high. Figure 4.7 shows graphs of the various sensations returned by the Zebra Zero during an insertion starting with the configuration shown in Figure 4.6.

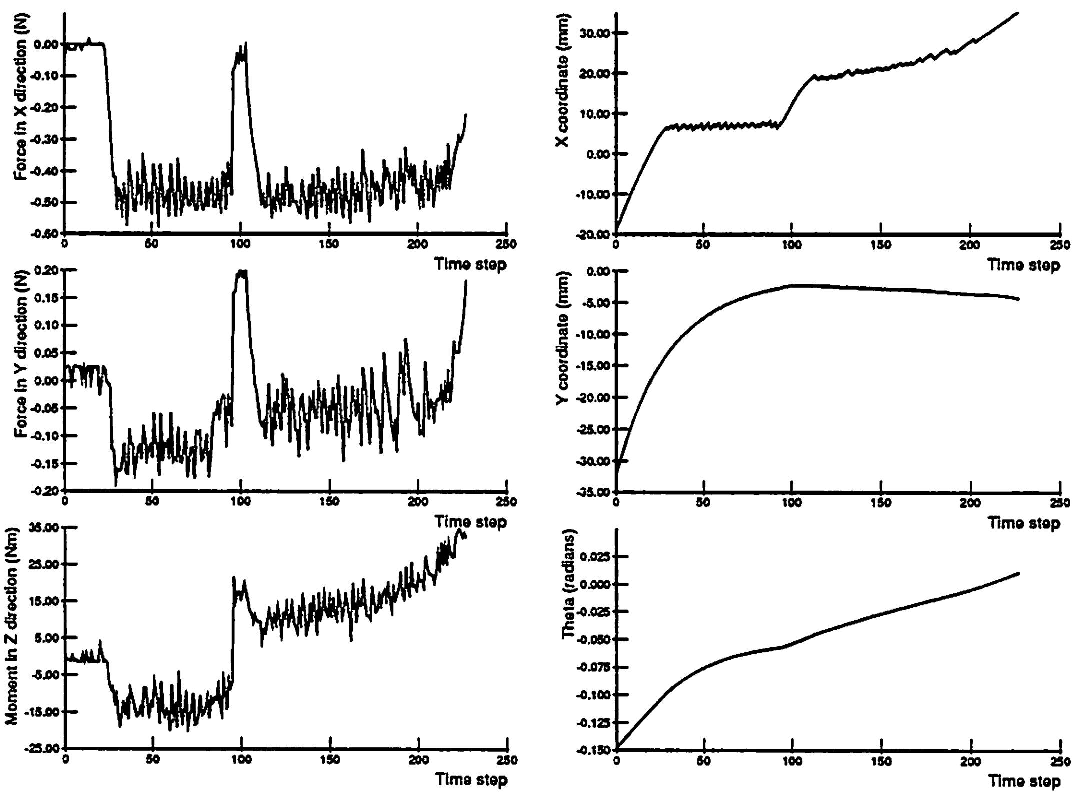


Figure 4.7. *Peg-in-hole task: Sensory feedback during the course of insertion for test case 6 in Table 4.1. These data were obtained after the controller had completed 100 training runs.*

Our results, as well as those of Lee and Kim, indicate that direct reinforcement learning can be used to learn a reactive control strategy that works well even in the presence of a high degree of noise and uncertainty. In contrast, the degree of uncertainty inherent in physical systems raises doubts about the practicality of a supervised learning approach such as that of Asada [9] described above. A complete case-by-case analysis of the statics of the peg-in-hole task can be used to derive situation-action rules that specify the corrective motion for a given force signature in a given position, which can then be used as training information for a supervised learning system. However, due to sensing noise, the position and force sensations in a given situation differ vastly in practice from those predicted by analysis. Moreover, in practice, the sensations are further corrupted by factors such as sampling rates and sampling times that come into play when interfacing the controller to a real system. Thus, for example, the force signature obtained from the physical system could differ substantially from that predicted by a static analysis simply because the system was not at a static equilibrium at the time the sensor values were sampled. Other factors used in the analysis, such as the coefficient of friction between the peg and the hole, could also differ from the true values for a particular physical realization of the task.

Unless the analysis used to obtain the situation-action rules takes these factors into account—which is easier said than done—a controller trained using these rules cannot be expected to perform well in practice. These same problems of noise and uncertainty also make modeling the physical system difficult, hindering the application of indirect methods. We therefore believe that in these situations it is easier to use direct reinforcement learning methods to learn a reactive control strategy that enables good performance despite the presence of uncertainty and noise.

4.2 Learning inverse kinematics of an under-constrained manipulator

The second task we present involves learning inverse kinematics in the presence of excess degrees of freedom. The task is a robot arm positioning task in which the controller has to move the end-effector of a 3-joint planar arm to one of three target locations. To do this, the controller has to solve an inverse kinematics problem, which is to determine appropriate joint angle commands that result in positioning the arm's end-effector at a desired location. The particular version of this task used is shown in Figure 4.8. The input to the controller is a command specifying which of the three square buttons to position the end-effector over. The actions of the controller are joint angle settings[2] that alter the arm configuration and hence the location of the arm's end-effector. Clearly, this task is an example of learning with distal targets because the targets are desired values of the Cartesian location of the end-effector and the latter is the output of a process (the robot arm) driven by the controller's actions.

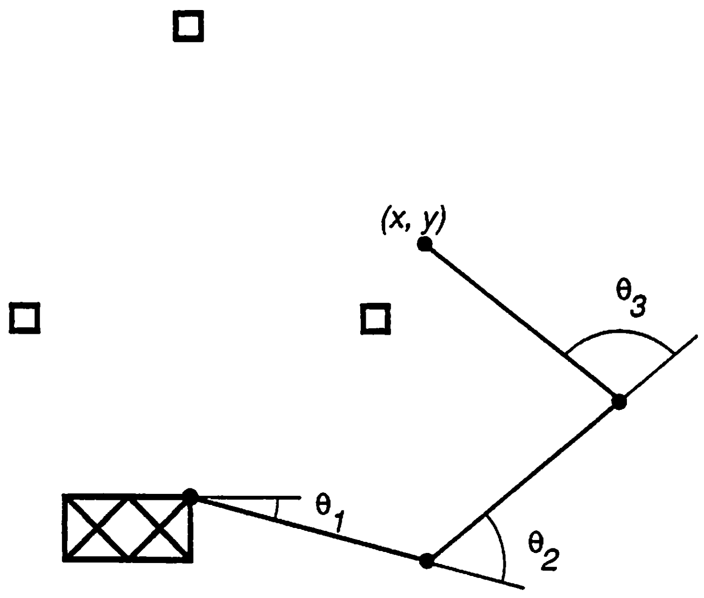


Figure 4.8. *The inverse kinematics task. Based on a command input that specifies which button, the network is to position the end-effector of the robot arm on one of the three buttons.*

A further complication exists here. Due to the presence of excess degrees of freedom in the arm for this task, there are no unique solutions: many different actions of the controller can result in the same end-effector location. This makes it difficult to specify desired actions a priori, and hence training information required for supervised learning cannot be easily obtained. Moreover, as discussed in Section 2.3,

[2]All the joints of the arm are revolute.

direct inverse modeling methods are also ineffective under such circumstances. The inverse kinematics problem for under-constrained manipulators is the subject of much robotics research even today.

The above version of the inverse kinematics problem was studied previously by Jordan and Rumelhart [68], who used an indirect method for solving it. Their approach involved constructing a differentiable forward model of the robot arm and using the forward model to obtain the gradient of an error function in Cartesian space with respect to the control actions. Jordan and Rumelhart also used knowledge of the kinematic equations of the robot arm to select the representation for the controller's actions. Because the joint angles of the arm appear only as sines and cosines in the forward kinematics equations, they used the cosines of the joint angles as the control actions, although the actual control variables are the joint angles themselves. This representation considerably eased the construction of a forward model of the robot arm.

Jordan and Rumelhart [68] used back-propagation networks to implement both the controller and the forward model. Their controller had 2 input, 50 hidden, and 3 output units and their forward model had 3 input, 50 hidden, and 2 output units. Figure 4.9a shows a block diagram of their architecture.

A fairly elaborate study of this problem was conducted by Jordan and Rumelhart [68], and one of points they make is that there is a trade-off between the amount of time spent in training the model network and the amount of time required to train the controller using the resulting model (see Figure 16 on page 27 in ref. [68]). They note that in applying indirect methods, often a very crude forward model suffices for solving the control problem. However, as illustrated by their results, using a crude model can also slow down control acquisition. Based on their simulations of the indirect method on this task, two points are worth noting here.

1) Although training the forward model for more than about 500 time steps, or trials, does not result in any significant savings in the amount of training required for

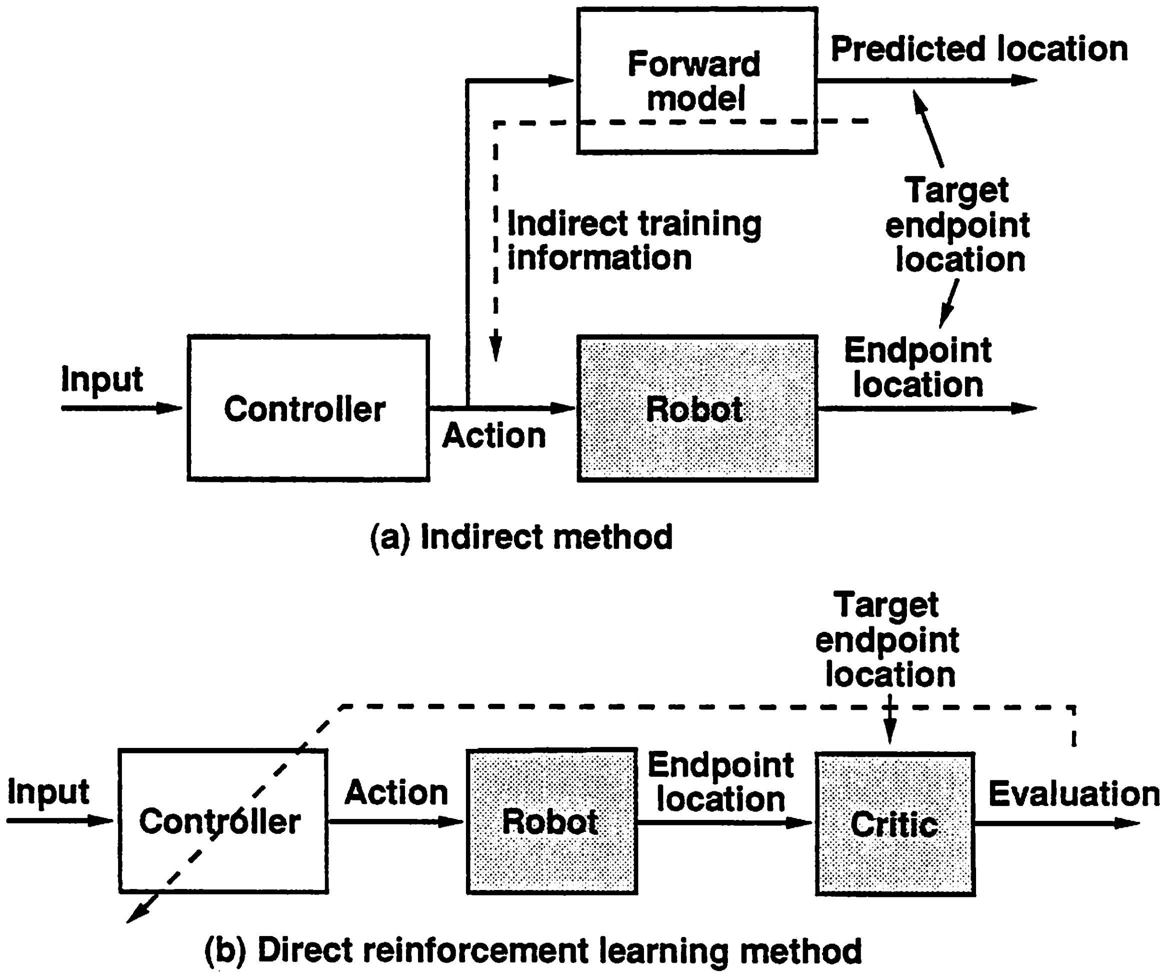


Figure 4.9. *The network architecture used for the inverse kinematics task. Part (a) shows the architecture used by Jordan and Rumelhart [68]; part (b) shows the architecture used with the direct learning method*

the controller on this task, such a limit on the accuracy of the model is usually not known a priori. This is in fact a significant limitation of indirect methods. Because the model accuracy needed for any particular task is not known in advance, one cannot minimize the time spent on training the model without actually using it to train the controller.

2) It is usually the case that training the controller and the model simultaneously leads to poorer learning performance than training the model is first and then training the controller. This is because with simultaneous training the controller starts off by learning incorrect actions based on training information obtained from the initially inaccurate model. This "bad" initial control knowledge interferes with subsequent acquisition of good control by the controller. Indeed, in most tasks, it is prudent to construct at least an approximate forward model before commencing training the controller. For the task under consideration, for example, Jordan and Rumelhart [68] find that prior training of the forward model for about 50 trials is essential to ensure that the training information derived from the model is useful for learning control.

Although in principle, one can interleave modeling and training the controller, the observations above indicate the difficulty of determining when such interleaving is advantageous.

Results and Discussion

Simulation results reported by Jordan and Rumelhart [68] indicate that the lowest *total* number of training trials for both the model and the controller is around 1250 trials.[3] We conducted some simulations using a direct learning method on this same task. The architecture used in these simulations is shown in Figure 4.9b. The

[3]None of the numbers quoted in this section should be taken as definitive because neither Jordan and Rumelhart nor we have attempted to optimize learning speeds through tuning parameters and network structures. The descriptions and results presented here are intended to highlight the qualitative aspects of using direct and indirect methods. We hope that this will motivate rigorous quantitative comparisons of the two methods.

controller network used is identical to the one used by Jordan and Rumelhart but with SRV units substituted for back-propagation units in the output layer. The evaluation function used by the critic was based directly on the error function used in the indirect method. In our simulations, the direct method learned to solve the task in 1269 trials on the average (over 20 runs). This performance is comparable to that of the indirect method and is obtained without the computational overhead of training and using a model.

Also recall that the task of learning a forward model was considerably simplified by the representation chosen for the control actions: Cosines of the joint angles were used as the control actions, although the actual control variables are the joint angles themselves. Because the mapping from cosines of joint angles to Cartesian locations of the end-effector is more nearly linear than the mapping from joint angles to Cartesian locations, learning a forward model becomes simpler. Obviously, such simplifying choices of representations are possible only if there is prior knowledge of the mapping to be learned. In the absence of such prior knowledge, the performance of the indirect method would have suffered because of the increased complexity of forward modeling. The direct method, however, does not have this drawback.

4.3 Learning to control an unstable dynamic system

The third and final set of experiments presented in this chapter involves learning to control an unstable non-minimum phase dynamic system, namely the much-studied two-dimensional cart-pole system [150, 92, 23, 125, 29, 7, 66, 151]. The cart-pole learning control task, also known as the pole-balancing or inverted pendulum problem, involves learning to balance a rigid pole mounted on a mobile cart by applying appropriate forces to the cart. As depicted in Figure 4.10, the cart is restricted to move along a one dimensional track of bounded length. The position and velocity of the cart $(x, \dot{x})$ and the pole $(\theta, \dot{\theta})$ represent the state of the cart-pole system.

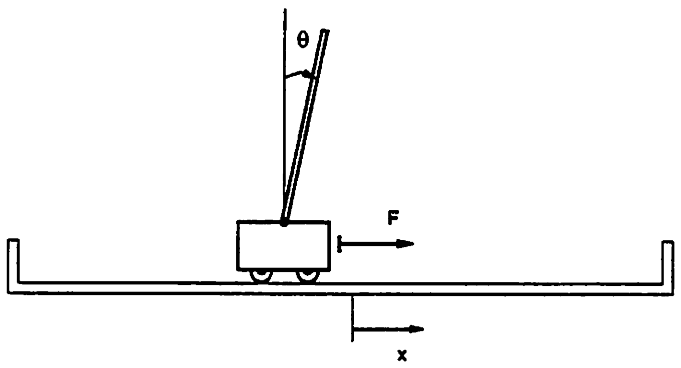


Figure 4.10. *The cart-pole system. x denotes the position of the cart on the track, θ denotes the angle the pole makes to the vertical, and F dentes the force applied to the cart.*

For this task, training information supplied to the controller is in the form of a failure signal, f_{ext}, that occurs when the pole falls beyond a critical angle or the cart hits either end of the track. Clearly, this training signal occurs infrequently, creating a difficult *temporal credit assignment problem* [136]. Approaches to this credit assignment problem usually involve training an internal critic network to generate predictions of future evaluations. In the absence of external evaluations, these predictions are used by the learning system as if they were actual evaluations received from the environment. Examples of algorithms used for learning to predict future evaluations include the TD(λ) (Temporal Difference) family of algorithms developed by Sutton [137] and the related Q-learning algorithm developed by Watkins [141].

As in the previous section, the performance of direct and indirect learning methods are compared on this task. The indirect approach, presented by Jordan and Jacobs [66], involves treating the internal critic as a "forward model" and using its *structure* to obtain training information for the controller. The direct method is based on reinforcement learning using the evaluations produced by the internal critic and has been implemented using the Jordan-Jacobs work as a reference, thereby facilitating a direct comparison.

For the cart-pole task, Jordan and Jacobs used the temporal difference algorithm outlined below to train the internal critic. The same algorithm has been used in the simulations reported here. The algorithm was designed to predict z_t, the *reciprocal* of the time until failure, given the current time step t. Hence z_t should equal 1 on the time step immediately preceding failure, and should also satisfy the relationship

$$\frac{1}{z_t} = \frac{1}{z_{t+1}} + 1.$$

Therefore the error function used to train the internal critic network to estimate the reciprocal of time until failure is:

$$\epsilon_{prediction}(\hat{z}_{t-1}) = \begin{cases} 0.5(1 - \hat{z}_{t-1})^2 & \text{if there is a failure at time } t, \\ 0.5\left(\frac{1}{1+1/\hat{z}_t} - \hat{z}_{t-1}\right)^2 & \text{otherwise,} \end{cases} \tag{4.1}$$

where $\hat{z}_t$ is the estimate produced by the internal critic. Moreover, the evaluation (failure signal) seen by the controller at each time step becomes

$$f_t = \begin{cases} f_{ext} & \text{if there is a failure at time } t, \\ \hat{z}_t & \text{otherwise.} \end{cases} \tag{4.2}$$

The network architecture used by Jordan and Jacobs is shown in Figure 4.11 (compare with the indirect control architecture shown in Figure 2.3b). Although it is very similar to some previously used architectures (e.g., [23, 7]), it includes connections (shown as bold arrows in Figure 4.11) through which the output of the controller is fed as input to the internal critic.[4] It is through these connections that error gradients are propagated back to serve as training signals for the controller. The error gradient was computed as follows. Because the goal of the controller is to avoid failure, an appropriate desired failure signal is zero (signifying no failure), and an appropriate distal error function for the controller is

$$\epsilon_{control}(f_t) = 0.5(0 - f_t)^2.$$

The corresponding distal error gradient for the controller, f_t, is back-propagated through the internal critic network and through the connections to the controller to

[4]In keeping with the terminology used in [68], Jordan and Jacobs [66] refer to the internal critic as the *forward model.*

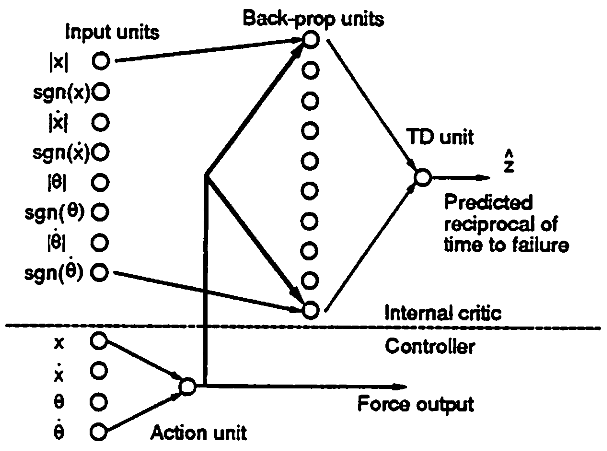


Figure 4.11. *The network architecture used for the cart-pole task. The internal critic network has 8 input units, a hidden layer of 10 back-propagation units, and a single temporal difference (TD) output unit. The controller has 4 input units and a single action unit. In simulations of the indirect method, a "noisy" linear unit was used as the action unit, while in simulations of the direct reinforcement learning method, an SRV unit [48] was used.*

provide a proximal error gradient for the controller. The weights of the controller are then adjusted to descend the proximal error gradient, which in turn results in minimizing the distal error and hence driving the failure signal to the desired value of zero. The internal critic network is also trained simultaneously with the controller. In its case, however, back-propagation is used to adjust the weights so as to minimize the *prediction error* defined in Equation (4.1).

The architecture of the network used with the direct method was identical to the one used with the indirect method except for the following difference (see Figure 4.11). The controller's noisy linear action unit was replaced by an SRV unit. For the direct method, training information for the controller is a reinforcement signal, r_t, that is computed directly from the internal evaluation signal f_t as

$$r_t = 1.0 - f_t.$$

Maximizing the expected reinforcement therefore results in the desired effect of minimizing the internal evaluation. The internal critic network is also trained simultaneously, using back-propagation to adjust the weights so as to minimize the prediction error (Equation 4.1), as with the indirect method.

Results

Our simulations of the indirect method are replications of those in Jordan and Jacobs [66], and we have tried to keep the differences between simulations of the indirect and direct methods to a minimum. One hundred training runs were performed using both the indirect and direct methods. Each training run represents an attempt to train the network, starting with small random initial weights, to balance the pole. As in ref. [66], a training run lasted until either the pole was balanced for 1000 consecutive time steps or 30,000 failures occurred during the course of the run. Following ref. [66], in all the simulations the controller produced real-valued forces, and we restarted the cart-pole system in a random state after each failure. We also used the input representations in ref. [66] (see Figure 4.11), which take advantage of the symmetries in the cart-pole system dynamics. Furthermore, the learning rate parameters used for the network using the indirect method were also identical to those used in ref. [66].[5] Instead of optimizing these parameter values for the network using the direct method, the values used with the indirect method were retained.

Additionally, in the simulations using the indirect method, we incorporated Jordan and Jacobs' assumption that disturbances in the form of additive noise having a uniform distribution on $[-0.001, 0.001]$ act on the actions of the controller. The role of these disturbances will be discussed later. In order to evaluate the effect of such disturbances on the direct method, we also ran 100 runs of the direct method with identical noise added to the actions of the controller.

When the indirect method was used, 72 out of 100 training runs resulted in success, with the remaining 28 failing to balance the pole on any of 30000 attempts.

[5]These learning rate parameters were 0.05 for the controller and 0.3 for the internal critic network except on the connections from the controller, where the learning rate was 0.9.

The average number of failures before success over all the successful training runs was 11164.5 failures, and the median was 9244 failures. In the case of the direct method, 93 training runs were successful, with an average number of failures before success of 2385.89 failures, and a median of 1148 failures. When disturbances were added to the output of the reinforcement learning controller, the number of successful runs dropped to 88. The average number of failures before success over all the successful runs was 2689.3 failures and the median was 1383.5 failures. Histograms of the distribution of the successful training runs in terms of the number of failures before success for each learning method are shown in Figure 4.12.

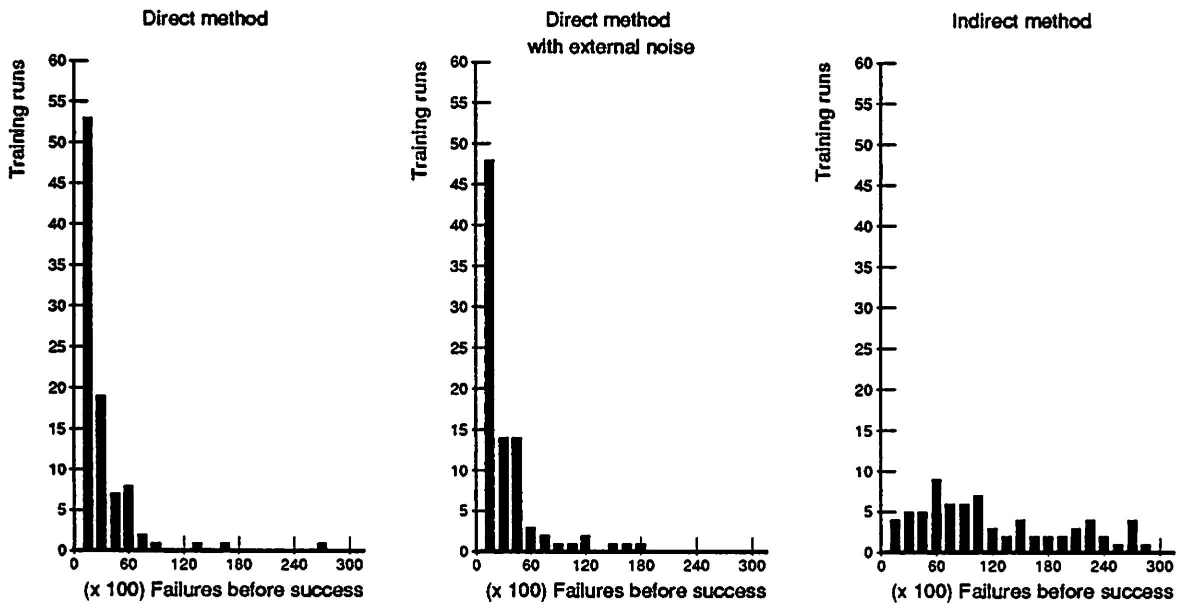


Figure 4.12. *Performance on the cart-pole task. The bar graphs show the distributions of the successful training runs in terms of the number of failures before success using each learning method.*

These results indicate a substantial difference between the two learning methods in terms of learning speed and overall learning behavior. Some reasons for these differences are discussed in the next section.

Discussion

One of the reasons that comparison of direct and indirect learning methods on the cart-pole task is particularly meaningful is the necessity of an internal critic for generating evaluations at each time step in order to solve the temporal credit assignment problem. The internal critic is necessary *regardless* of which of the two learning methods is employed and has to be constructed through training experience. Whereas both the direct and indirect methods need the internal critic to obtain evaluations in the absence of the environmental failure signal, the indirect method also uses the *structure* of the internal critic network to translate (via back-propagation) distal errors in evaluations into proximal errors in actions of the controller. Therefore the difference in performance of the two methods essentially arises from differences in the accuracies of the internal critics constructed during training and the manner in which they are used, and *not* from any bias in the task setup or the training information received.[6]

Let us first consider the problem of constructing an adequate internal critic. If the internal critic is to produce accurate evaluations over all of its input space, it is vital that it "sees" inputs that are distributed over its entire input space. In other words, exploration of the (state, action) input space is imperative. The two methods under consideration rely on different mechanisms to effect this exploration. In the indirect method, external disturbances in the form of additive white noise are assumed to affect the control output *at all times*, perturbing the actions of the controller and hence introducing variations in the inputs to the internal critic. A potential problem with this approach is the lack of direct control over the noise and hence over the exploration. While higher magnitudes of noise will enable better exploration—leading to a more accurate internal critic and thus to faster control acquisition—large-magnitude noise

[6]If, for example, the environment provided appropriate evaluations at all times, then the direct method would not be required to construct an internal critic, while the indirect method would still have to do so. Such a situation would therefore be biased in favor of the direct method.

is detrimental to the long-term performance of the controller. Conversely, lower noise levels will facilitate good control after learning but will slow the learning of a useful internal critic and hence slow the acquisition of control. Unfortunately, because the source of the noise is external to the learning system, its amplitude is clearly beyond the control of the learning system.

In contrast, active exploration of the action space is an integral feature of direct methods. In order to discover the best output to produce, direct methods generate actions stochastically and use the resultant evaluations to guide the search for the best output. On the cart-pole task, using reinforcement learning therefore facilitates the desired exploration of the input space of the internal critic. Moreover, direct methods must directly control the randomness in the output in order to exhibit convergent learning behavior. This means that as learning proceeds and performance on the task improves, the exploratory variations in the controller's output decrease to zero.

The manner in which the internal critic is used to train the controller also has a critical impact on the learning behavior. As discussed above, the indirect method is based on the idea of using the structure of the internal critic to obtain gradient information for training the controller. Jordan and Rumelhart [68] show how the back-propagation process computes the gradient of the output of the internal critic with respect to the controller's output. If the controller's training information is to be meaningful, the internal critic has to be reasonably accurate at the current point in (state, action) space, *and* the gradient computed using the internal critic has to have a positive inner product with the true gradient of the output of the internal critic with respect to the controller's actions (see ref. [68], pp. 11-13). We have already discussed the factors governing the first condition. The gradient, however, is defined mainly by the weights on the connections from the controller to the internal critic. Therefore the indirect method is critically dependent on the weights on these connections. In other words, the *structure* of the model becomes critical when indirect methods are used.

But because of the redundancy of the information provided to the internal critic via these connections,[7] the internal critic can make accurate predictions even with arbitrary weights on these connections. Therefore, in practice, there is reason to believe that the predictions of the internal critic become accurate long before these weights take on meaningful values. This is because accurate predictions are possible independently of these weight values, and once the predictions are accurate, there is no output error to drive the adaptation of these weights. Fully aware of this problem, Jordan and Jacobs [66] sought to mitigate its effect by increasing the learning rate on the connections from the controller to the internal critic. Nonetheless, due to this problem, the provision of meaningful gradient information to the controller is delayed, thereby slowing learning.

In comparison, the direct method relies only on the *output* of the internal critic for its training information. Because the reliability of the predictions of the internal critic is probably higher than that of the gradient estimates obtained using the internal critic's weights, the direct method gets meaningful training information more frequently than the supervised learning method. The drawback of using direct reinforcement learning, however, is that the gradient of the output of the internal critic with respect to the controller's output has to be estimated stochastically. Nevertheless, the results presented above indicate that estimation of the gradient by the direct method can be more accurate than estimation of the gradient through the weights of the internal critic.

The empirical results reported in this chapter illustrate the power and utility of direct reinforcement learning methods for learning control. We have also compared the effectiveness of direct methods to that of indirect and other methods when applied to different kinds of control problems. In the next two chapters, we address the

[7]Providing the controller's output to the internal critic is redundant because the controller's output is a function of the state of the cart-pole system, which is already provided as input to the internal critic.

difficulties in scaling direct reinforcement learning methods to more complex control problems.

Chapter 5
Structural Credit Assignment

This chapter begins with a description of the problem of credit assignment as it appears in reinforcement learning systems. Both temporal and structural credit assignment problems are described, although our focus in this dissertation is on the latter. The structural credit assignment problem is defined in detail, and previous approaches to solving the problem in connectionist learning systems are discussed. Following this, we describe a novel modular connectionist architecture that can be used for structural credit assignment in a reinforcement learning system. Finally, we present empirical results demonstrating the utility of this architecture.

5.1 Credit assignment in reinforcement learning systems

The basic *credit assignment problem* for reinforcement learning systems was discussed by Minsky [96]. When sequences of decisions are made by various components of a reinforcement learning system before the system receives any evaluation from the critic, it is faced with the problem of assigning appropriate credit or blame to the various decisions. For example, a system learning to play checkers has to make a number of moves before the ultimate outcome (win, lose, or draw) becomes apparent. The system then has to decide which of the moves it made were really responsible for the outcome and which of its components were involved in generating those moves. There are two distinct aspects of the credit assignment problem called the *temporal* and *structural* credit assignment problems [136].

Temporal credit assignment—The temporal credit assignment problem arises because evaluations of the learning system's decisions may be delayed in time as in the checkers example given above or in the cart-pole task of Section 4.3. In the checkers example, a move or sequence of moves early in the game might have set up the board for the ultimate win, but it is difficult for the learning system to identify such moves. A commonly used approach to solving the temporal credit assignment problem is to develop a means of reliably predicting the future worth of a decision and to use the predicted worth to guide the learning process. This is the approach taken, for example, in Section 4.3 in training the controller in the cart-pole task.

The checkers-playing program written by Samuel [119, 120] also used a method for adjusting the coefficients of the terms in an evaluation function so that the value assigned to a board position incorporated the worth of board positions reachable in several moves. More favorable evaluations were thus assigned to board positions that led to good positions later in the game, enabling reliable selection of good moves early in the game.

The Adaptive Critic Element (ACE) algorithm developed by Sutton [136] is a method for learning to predict future evaluations that is related to Samuel's method. This algorithm was incorporated into a connectionist architecture called the Adaptive Heuristic Critic (AHC) [23]. Sutton [137] extended the ACE algorithm into a family of algorithms called the Temporal Difference (TD) algorithms, which have been shown to be related to dynamic programming by Barto, Sutton, and Watkins [25]. Several other temporal credit assignment methods have been studied by researchers (see, for example, Holland [59], Werbos [144], and Watkins [141]). Although we consider the temporal credit assignment problem to be important, our study focuses on the structural credit assignment problem.

Structural credit assignment—The problem of structural credit assignment arises when the learning system's actions are based on the actions of more than one constituent component of the learning system. In such cases, the evaluation received by the system for an action has to be correctly apportioned between the contributing components. The structural credit assignment problem has two aspects. The "hidden component" credit assignment problem involves assigning credit to those components of the learning system that do not directly interact with the environment. The "multiple action element" credit assignment problem, on the other hand, arises when the learning system's actions are multi-dimensional. Here, the learning system has to determine the relative impact of each action element in various situations to apportion credit among the action elements for the ensuing evaluation.

The significance of both aspects of the structural credit assignment problem is readily apparent in reinforcement learning systems. Because usually only a scalar evaluation is returned by the critic, credit has to be properly apportioned between all the action elements *and* all the hidden components of the learning system based on just this scalar value. However, even supervised learning systems must deal with one aspect of structural credit assignment. Although the teacher assigns credit among multiple action elements in supervised learning systems (by providing an individual target value for each action element), the problem remains of assigning credit to the hidden learning components (e.g., the hidden units in a connectionist network). This problem has received considerable attention from connectionist researchers because of its centrality to training multilayer networks. Indeed, much of the interest in connectionist learning can be attributed to the development of techniques for solving the hidden component credit assignment problem in multilayer networks.

Techniques for hidden component credit assignment in supervised learning systems can also be used in reinforcement learning systems. However, additional techniques are needed for credit assignment among multiple action elements of a reinforcement learning system. In the sequel, we present a novel connectionist architecture that can be used to assign credit to individual action elements of a reinforcement learning controller. Before doing so, however, we review existing approaches to structural credit assignment in connectionist systems.

5.2 Some methods of structural credit assignment

In his PhD thesis, Anderson [6] presents an extensive review of various approaches proposed for structural credit assignment in connectionist networks. He classifies the approaches into three main groups based on the general approach taken: (1) gradient methods, exact and approximate; (2) methods based on a minimal-change principle; and (3) methods based on a measure of worth of a network component. We briefly discuss the prominent methods from each group here and refer the reader to Anderson's thesis for further details.

5.2.1 Gradient methods, exact and approximate

Gradient methods are the most popular means of solving the structural credit assignment problem. They involve computing the exact or approximate gradient of a criterion function with respect to the outputs of each of the units in the network. The gradients indicate the degree of influence of each unit on the criterion function, and hence each unit is assigned credit proportional to the magnitude of the gradient. Backpropagation (e.g., [117]) is an example of a supervised learning algorithm that computes the exact gradient of a per-sample error function (defined over the network's outputs) with respect to each hidden unit's output. It is therefore a method for structural credit assignment among the hidden units in a multilayer network. Indeed, the error gradient propagated back to each hidden unit in the backward pass of the algorithm represents the fraction of the per-sample error ascribed to that unit.

Other supervised learning algorithms use estimates of gradients to assign structural credit to the hidden components. Examples include the Boltzmann machine algorithm [1], and the *backpropagating error correction procedure* of Rosenblatt [114], which is similar to backpropagation but relies on a probabilistic estimation of the error gradients instead of computing the exact gradient.

Several reinforcement learning systems also rely on estimates of gradients as a means of credit assignment.[1] An early example is the SNARC (Stochastic Neural-Analog Reinforcement Calculator) system described by Minsky [95] in his PhD thesis. Farley and Clark [38] developed a method for modifying the weights of randomly connected stochastic units based on correlations between the outputs of the units and the evaluation. As discussed in Chapter 3, many other similar algorithms have been studied by learning automata theorists [104] and animal learning theorists [73, 74, 138]. Much of the work along these lines has been done by Barto and his colleagues [24, 22, 23, 136, 18, 15, 19, 7], who have developed and studied several algorithms based on these ideas. Prominent among these is the A_{R-P} algorithm of Barto and Anandan [18]. In our own work, we have shown that under certain conditions, the SRV unit algorithm uses unbiased estimates of the gradient of the expected evaluation to update the unit's weights (see Chapter 3 and Lemma 2 in Appendix A).

5.2.2 Methods based on a minimum-change principle

Minimal-change methods use various heuristics to determine how any particular unit in the network affects the overall error or the overall evaluation. The heuristics are designed to detect a minimal set of units such that altering their weights eliminates the error in the network output. Some of these methods rely on direct observations of changes in the error with changes in a unit's weights. Examples of such methods include the procedure developed by Widrow [147] for adjusting the weights of networks of Adalines (Adaptive linear elements), and Stafford's methods [133, 134] for adapting the weights of hidden units in networks with a single output unit. Other minimal-change methods are based on the notion of a *prototype* unit.

[1]It is perhaps more accurate to say that in reinforcement learning, *correlations* between the outputs of units and the reinforcement, or evaluation, are used to assign credit to the units. However, it can be shown that for most reinforcement learning algorithms, the correlations formed are estimates of the gradient of the expected evaluation with respect to the outputs of the units (see [104, 18, 152, 48] and Appendix A).

Here, units are assigned credit based on the degree of match between their weights and the input vector. Typically, the weights are regarded as defining the center of a hypercube or hypersphere, and the distance between the center and the input vector becomes a measure of the match. Only the weights of the units with the highest degree of match are adjusted by the learning algorithm. Several such methods have been described (see, for example, Skolic[2] [130], Hampson [52], Rumelhart and Zipser [118], Moody and Darken [98], and Niranjan and Fallside [108]).

5.2.3 Methods based on a measure of worth of a network component

In these methods, various measures of worth are used to determine how to streamline the structure of the learning system by discarding useless components and by adding useful new components. The worth of a component is estimated by evaluating the contribution it makes to the output of the learning system. A commonly used measure is the overall output weight of a unit. (The overall output weight of a unit is the sum of the magnitudes of the weights on the connections between the output of the unit and other units.) This measure was used, for example, by Samuel [119] in his checkers-playing program, by Selfridge [123] in Pandemonium, and by Holland [59] in his bucket-brigade algorithm. Other measures of worth include the product of the unit's output and its overall output weight and the cross correlation between the unit's output and the desired output. On comparing these three measures of worth in a two-layer connectionist network with a single output unit, Klopf and Gose [75] found the best among them to be the product of a unit's output with the overall output weight of the unit.

The cascade-correlation technique of Fahlman and Lebiere [37] uses a novel measure of worth to add new hidden units to a network so as to improve performance.

[2]Skolic actually presents his method as a decision making procedure in a production system, but his methods can be recast into the connectionist framework.

Each unit in a pool of candidate units is trained to maximize the correlation between its output and the errors produced by the existing network. The candidate units receive inputs from all input and hidden units in the existing network. The best candidate unit, i.e., the unit with the maximum correlation, is then added as a hidden unit in the network, and the network is retrained. Adding the new unit is beneficial because it essentially acts as a detector for error-causing inputs, and hence the unit's output can be used to cancel out the errors.

Several other researchers have developed methods to *prune* networks based on various measures of worth. Examples include Rumelhart's method of adding extra cost terms to the error function minimized by backpropagation [115], used by Hanson and Pratt [55], the "optimal brain damage" technique of le Cun et al. [81], and the skeletonization technique described by Mozer and Smolensky [99].

5.3 A specific gradient method: Differentiating a forward model

In the previous section, we described the utility of gradient based techniques, such as backpropagation, for hidden component credit assignment in supervised learning systems. In this section, we shall describe how these techniques can be used for credit assignment among multiple action elements of a controller as well. The discussion in this section will serve as a background for presenting our architecture for multiple action element credit assignment in a controller trained using reinforcement feedback.

In Section 2.3.1, we presented Jordan and Rumelhart's [64, 68] approach to learning with a distal teacher as an example of the indirect approach to learning control. The description presented in that section was from the point of view of translating distal training information into a proximal form useful for training the controller. Recall that the method involves constructing a forward model of the process and training the controller by using the forward model to translate error gradients in the output space of the process into error gradients in the action space of the controller. An alternative point of view of this method is presented by Barto [16], who points out that generating error gradients for the controller's actions given distal training information is essentially the same problem as obtaining error gradients for hidden units in a network given the desired network outputs. In both cases, the gradients are not directly available and must be computed in some fashion. Jordan and Rumelhart's [64, 68] approach is an elegant method for computing these gradients. By appending to the controller network another network that is a model of the controlled process as shown in Figure 5.1, the output layer of the controller is transformed into a hidden layer in the overall network. Thus, the problem of credit assignment among multiple action elements is transformed into a hidden component credit assignment problem, which can be solved using available techniques from supervised learning such as backpropagation.

A similar indirect method proposed for training controllers in reinforcement learning tasks [100, 66, 145] was also described in Section 2.3.1. An implementation of this approach by Jordan and Jacobs [66] was described in detail in Section 4.3. Other implementations of this approach include those of Munro [100] and Bachrach [12].

From the above it is clear that one of the advantages of using an indirect method for training the controller is that it can solve the multiple action element credit assignment problem. In the next section, we elaborate this approach and present a variation of this idea that might be more practical for reinforcement learning tasks.

5.4 Using consequent process output for credit assignment among multiple action elements

The focus in this section is on the multiple action element credit assignment problem in a reinforcement learning system. When a multi-element control action is performed, the controller has to determine which action elements were, or were not, responsible for the ensuing evaluation. While the standard reinforcement learning approach relies on correlations between perturbations of action elements and

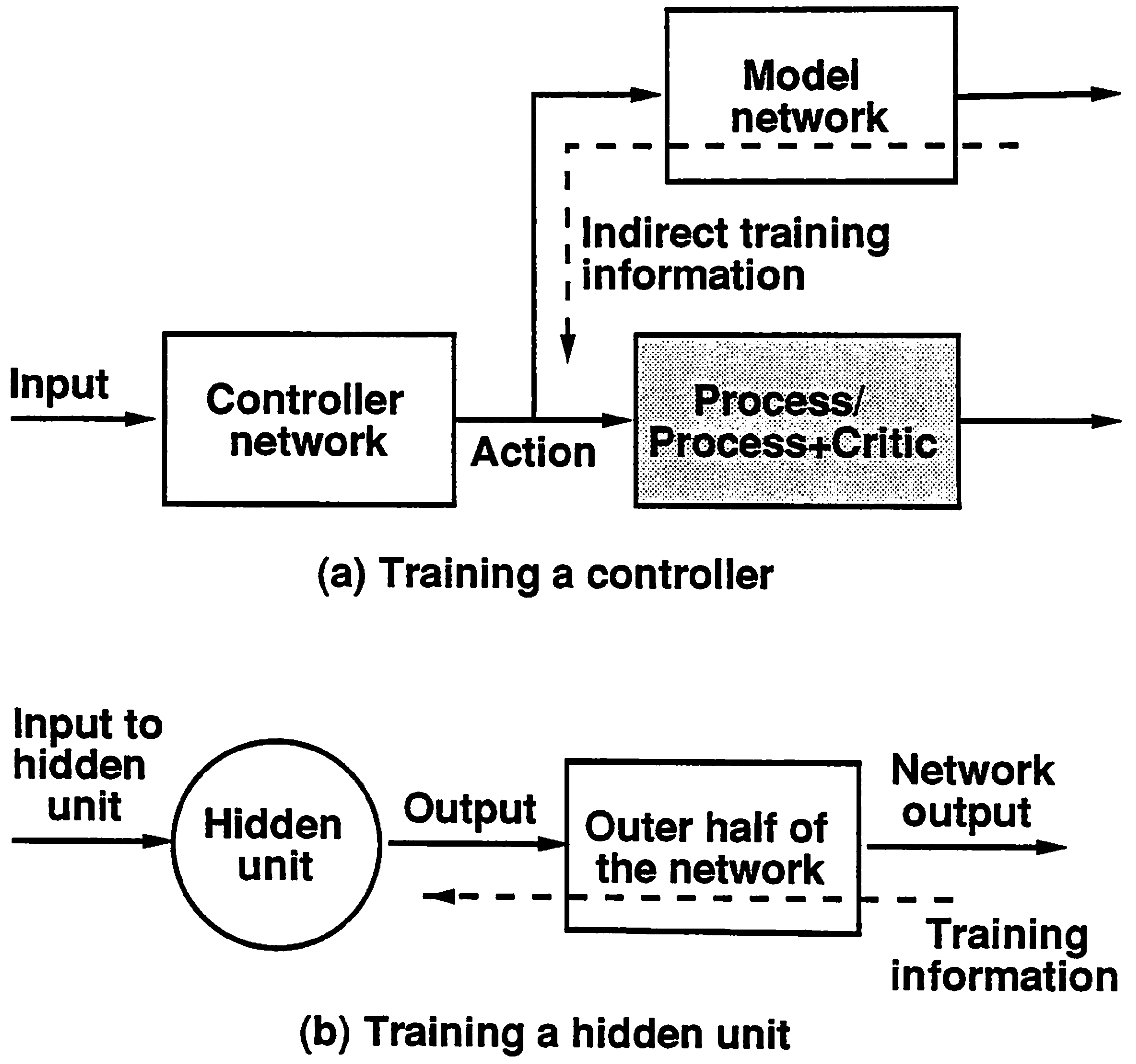


Figure 5.1. *Training a controller using indirect methods is like training a hidden unit in a network. After Figure 4 in Barto [16].*

consequent changes in the evaluation (as described in Chapter 3), it is clear that such an approach is prone to error. Even though perturbing some action elements might not have affected the ensuing evaluation, a reinforcement learning method may detect spurious correlations between these perturbations and the observed changes in the evaluation.[3] In other words, standard reinforcement learning methods cannot distinguish between correlations arising from *contingency* of the evaluation on the action elements and those due to accidental *contiguity* between perturbations in action elements and changes in the evaluation.

Modification of control behavior based on spurious correlations leads to what experimental psychologists studying reinforcement learning in animals call *superstitious* or *adventitious* learning [128, 132]. Superstitious learning can be detrimental to learning control because adjustments to an action element based on spurious correlations can disrupt adjustments made using correlations observed when the evaluation is really contingent on that action element. As the number of elements constituting a control action increases, the likelihood of spurious correlations between action element perturbations and changes in evaluation also increases, making superstitious learning more likely. It is therefore necessary to devise a means of overcoming superstitious learning in a reinforcement learning system.

One approach to mitigating the deleterious effects of superstitious learning is suggested by the following observations. The state of the process, specified by the input to the controller, defines the *context* in which actions are generated by the controller and evaluated by the critic. Depending on the context, only a subset of the action elements might actually influence the evaluation, either because the rest of the action elements do not have a significant impact on the process or because their effect on the process is irrelevant for the task at hand. By identifying this subset of

[3]Mainly because most reinforcement learning methods use only a small number of observations to form correlations.

active action elements and assigning credit to only these elements, we can prevent superstitious learning by the inactive action elements.

We have already seen how the indirect method of constructing a global model of the evaluation function addresses this problem. But as argued below, this is not the only alternative, nor is it the best approach in all circumstances. For example, when some form of information about the relative influence of action elements is available, using it to directly assign credit among the action elements might be more expedient. One such source of information available in control tasks is the output of the controlled process, which reflects the effect of a control action on the process.

5.4.1 The heuristic

The idea underlying the credit assignment scheme presented here is to use information about the consequences of control actions on the controlled process to obtain a *sensitivity* measure for each action element. On considering the usual input–action–output–evaluation sequence in reinforcement learning control tasks (see Figure 2.2c), we see that the effect of the control actions on the evaluation returned by the critic can be regarded as a two-stage operation. First, the control actions operate on the process, causing changes in the process output. The critic then evaluates the changes in the process output within the current context (specified by the controller's input). Therefore, a tenable heuristic is that actions that have a small influence on process output in a given context also probably have a small influence on the ensuing evaluation. Based on this heuristic, we can assign a higher sensitivity rating to those actions that have a significant impact on the process output in a given context and a lower one to those that have very little impact.

Mathematically, we want to estimate the magnitude of the gradient of the evaluation function with respect to each action element, i.e., $\left|\frac{\partial r}{\partial y_i}\right|$, where r is the reinforcement, or evaluation, and y_i, $1 \le i \le n$, are the n action elements. The discussion in the last paragraph is based on the observation that

$$\begin{aligned}\left|\frac{\partial r}{\partial y_i}\right| &= \left|\sum_{j=1}^{m}\left(\frac{\partial r}{\partial z_j}\right)\left(\frac{\partial z_j}{\partial y_i}\right)\right| \\ &\le \sum_{j=1}^{m}\left|\left(\frac{\partial r}{\partial z_j}\right)\right| \left|\left(\frac{\partial z_j}{\partial y_i}\right)\right|,\end{aligned}$$

where z_j, $1 \le j \le m$, are elements of the process output vector. The credit assignment heuristic can therefore be restated as follows: If the second term, $\left|\frac{\partial z_j}{\partial y_i}\right|$, in each product on the right hand side of the above inequality is very small, the magnitude of the overall derivative, $\left(\frac{\partial r}{\partial y_i}\right)$, is also very small. Therefore, in a given context, little credit should be assigned to action elements y_i for which $\left|\frac{\partial z_j}{\partial y_i}\right|$ is small for all z_j, $1 \le j \le m$.

Rather than performing accurate credit assignment among the action elements, this heuristic is mainly useful for detecting action elements that have little or no impact on the process in a given context and can hence be considered inactive in that context. Isolating these action elements by assigning them low sensitivities in situations where they are inactive attenuates superstitious learning, thereby minimizing interference with learning when these elements *are* active. A limitation of this heuristic, however, is that it does not prevent superstitious learning in action elements that have a significant impact on the process but should nevertheless be considered inactive because their effect on the process is not relevant for the evaluation.

Also note that when $\left|\frac{\partial z_j}{\partial y_i}\right|$ is large but $\left|\frac{\partial r}{\partial z_j}\right|$ is small for all $z_j, 1 \le j \le m$, credit is wrongly assigned to the action element y_i because the heuristic considers the former term alone in assigning credit. But $\left|\frac{\partial r}{\partial z_j}\right|$ is small for all $z_j, 1 \le j \le m$, only when all the process output variables, z_j, are irrelevant for computing the evaluation. Therefore this heuristic might perform poorly if only process output variables that are irrelevant for computing the evaluation are used for credit assignment.

Summarizing, the idea is to use information in the process output to assign sensitivity values to individual action elements in each context. These sensitivity values are used to scale the learning rates when adjusting the action elements. Action elements with low sensitivity values undergo relatively smaller adjustment than action elements with high sensitivity, thereby reducing superstitious learning in the elements that are inactive in a given context.

5.4.2 Implementation of the heuristic

As indicated in the previous section, the quantities that need to be estimated for computing the sensitivities of the action elements are the partial derivatives $\frac{\partial z_j}{\partial y_i}$. These derivatives can be estimated in two ways. The first is to construct a global differentiable model of the process and evaluate the Jacobian of the model at the current output of the controller. This is very similar to the indirect method of using a process model to train a controller using distal training information as described in Section 2.3.1. An alternative approach that is more in the spirit of direct reinforcement learning methods is to construct a local linear model (LLM) of the process on-line and use it to estimate the required derivatives. Such models are easy to obtain and directly yield the desired Jacobian. Moreover, whereas global models can be characterized as being approximately correct everywhere, local models can be much more accurate, albeit only locally. Furthermore, stochastic direct reinforcement learning methods only require a local model that is valid within the "locality of search" defined by the range of action perturbations. The use of LLMs therefore fits neatly with the use of direct methods for training the controller.

The proposed architecture for implementing the credit assignment heuristic described above is shown in Figure 5.2. Although similarities to the architectures for implementing indirect methods (Figure 2.3) are obvious, there are several significant differences. First, the process model has only a single layer of linear units. Therefore, the output of the model, namely the predicted process output $\hat{z}$, is computed as

$$\hat{z} = W_{\text{LLM}}^{\mathsf{T}} y, \qquad (5.1)$$

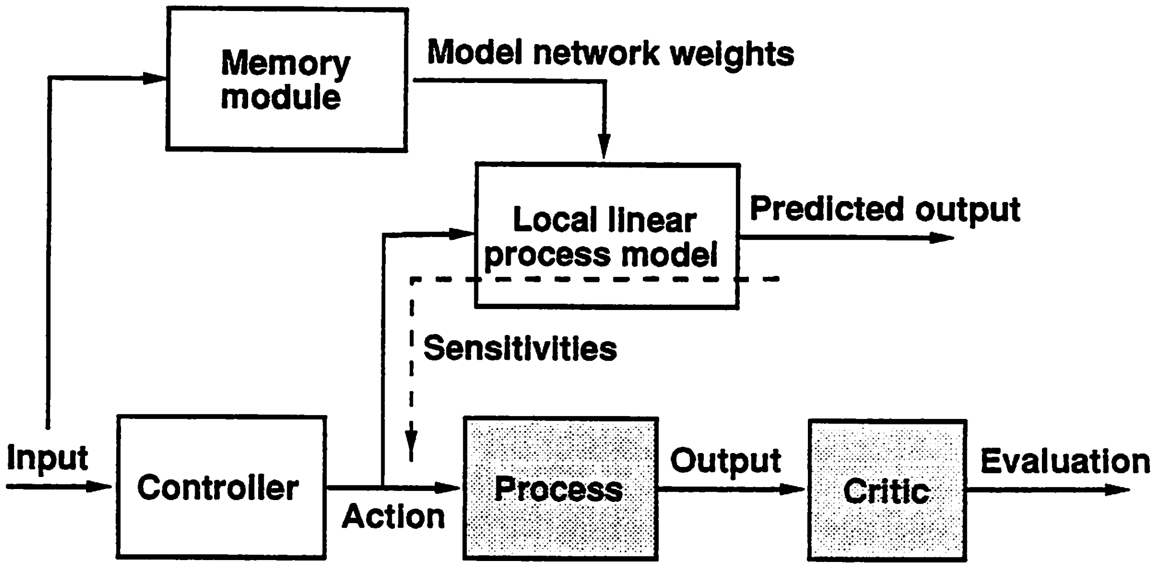


Figure 5.2. *An architecture for implementing the multiple action element credit assignment heuristic.*

where W_{LLM} denotes the matrix of weights of the LLM and y is the action of the controller. As noted above, one advantage of using a LLM is that the weights, W_{LLM}, can be used directly as estimates of the gradients $\frac{\partial z_j}{\partial y_i}$ needed for computing the sensitivities. On differentiating equation (5.1), we see that the weight matrix is an estimate of the Jacobian of the process at the current operating point. That is, the element w_{ij} of W_{LLM} is an estimate of $\frac{\partial z_j}{\partial y_i}$.

Let

$$c_i = \sum_{j=1}^{m} |w_{ij}|, \; 1 \leq i \leq n.$$

c_i is a measure of the net influence of action element i on the process output. The sensitivity, s_i, of the i^{th} action element is computed by normalizing c_i as follows:

$$s_i = \frac{nc_i}{\sum_{k=1}^{n} c_k}, \; 1 \leq i \leq n. \tag{5.2}$$

Normalization is used to determine the *relative* sensitivities of the action elements.

Clearly, the effectiveness of scaling the learning rate associated with each action element by the corresponding sensitivity value depends on the accuracy of the gradient estimates (i.e., the weights of the LLM) used to determine the sensitivity values. Inaccurate gradient estimates can cause credit assignment to the wrong action elements, which might actually increase superstitious learning. It is therefore safer in practice to assign equal credit to all the action elements while the weights are being estimated. The prediction error of the LLM provides a convenient measure of the accuracy of the model and can be used to determine when the information from the LLM is useful for credit assignment. Specifically, the sensitivities can be computed as

$$s_i = \begin{cases} \frac{nc_i}{\sum_{k=1}^{n} c_k} & \text{if } \|\mathbf{z} - \hat{\mathbf{z}}\| \leq \eta, \\ 1 & \text{otherwise,} \end{cases} \tag{5.3}$$

where η is a pre-specified limit on the prediction error.

From an implementational point of view, we note that although the weights, W_{LLM}, representing the process Jacobian in a given context can be learned incrementally through interactions with the process, it is possible to accelerate learning by using a *memory module* to associatively store the process Jacobian for each context. As depicted in Figure 5.2, the weights recalled by the memory module for any given context input can be loaded into the LLM. Although the memory module and the LLM together constitute a global model of the process, structuring the global model in this fashion enables fast computation of an estimate of the process Jacobian as the controller's context input changes, while at the same time preserving the advantages of using LLMs.

5.4.3 Relationship to "differentiating a model"

The LLM approach to multiple action element credit assignment in a reinforcement learning system is closely related to the approach described in Section 5.3 of differentiating a global model of the evaluation function. Both approaches are based on constructing and using a differentiable model in the manner of indirect methods, but they differ as to the kind of model constructed and the method used for constructing it. These differences are important in determining the relative efficacies of the approaches.

The first difference between the two approachs concerns what is modeled. The LLM approach requires constructing a model of the process alone, whereas modeling the evaluation function requires modeling both the process and the critic. Constructing and using a model of the process alone can be more practical and efficient for several reasons. Whereas a scalar evaluation combines the effects of all the action elements, a vector-valued process output is available in most control tasks, and each process output variable reflects the effect of a subset of the action elements. Therefore the process output conveys information that is useful for credit assignment among action elements, and the LLM approach makes full use of this information. Moreover, while the process characteristics are usually stationary in most learning control tasks, the evaluations depend on the performance of the controller, which is evolving over time, and the evaluation process itself can be non-stationary (for example, when *shaping* is used (see Chapter 6)). Modeling the process can therefore be easier than modeling the evaluation function. Finally, the same process model can be used for learning several different tasks involving different evaluation functions.

A second difference between the LLM approach and global modeling of the evaluation function is that unlike the latter approach, the LLM approach requires learning only a local model. We have already discussed the problems in constructing global models in Section 2.3.3. Learning an accurate local model of the process, however, can be done efficiently. For example, we can use proven techniques such as Recursive Least Squares [84] to learn a local linear model very quickly. Moreover, in our method, modeling effort is expended only in the region of action space where the controller is searching for the best actions, making modeling more goal-directed. As opposed to this, with global models, much effort may be expended in trying to learn an accurate global model in regions of the action space that will never be visited while learning the specified control task. The use of local process models also has the implementational advantages discussed in Section 5.4.2.

Finally, we observe that there exists a continuum of methods between the LLM approach and global modeling of the evaluation function. As we move along this continuum, the models become progressively more complex (linear to highly non-linear models and local to global models) and complete (models of the action-evaluation map instead of just the action-output map). Moreover, models could also have several intermediate stages with information injected into the modeling process at each stage.

An example in which a model has intermediate stages is the three-net architecture proposed by Werbos [145]. This architecture uses a two-stage model of the evaluation function, where the first stage is a forward model of the process and the second stage is a model of the critic. Werbos suggests that modeling the evaluation function in two stages and learning each stage based on signals from the environment might be more efficient and might lead to a more accurate model of the evaluation function. This is because the additional information in the process output can constrain the modeling process. Moreover, Bachrach [12] points out that the three-net architecture can be advantageous when the forward model of the process is known a priori because then only a model of the critic has to be learned.

In contrast, Jordan and Rumelhart [68] make the point that as far as training the controller is concerned, it may not be necessary to represent the dynamics of the process by a model. Instead, it may suffice to construct a simple global model from actions to evaluations that can "generalize suitably" over the space of actions, and then use this model to train the controller.

As can be seen from this discussion, there are many arguments to be made for and against the various approaches developed for addressing the structural credit assignment problem. The utility of any one approach clearly depends on the problem at hand. However, researchers interested in applying learning methods to control problems need to be aware of the wide range of methods that are available and the relationships between these methods. We hope that the above discussion contributes to this awareness.

5.5 Efficacy of the multiple action element credit assignment method: An illustrative example

In this section, we present simulation results that illustrate the utility of the approach presented in Section 5.4 for multiple action element credit assignment in a controller that uses direct reinforcement learning. The task used in these simulations is an abstract reinforcement learning task that has been selected specifically because it captures essential aspects of the multiple action element credit assignment problem. We describe the task first and then present the results of the simulations.

For reasons that will become clear shortly, the abstract reinforcement learning task used in this section is called the "interference task". This task can have an arbitrary number of action elements. For the general case of n action elements, the task is defined by a table of inputs and optimal actions, as shown in Table 5.1. A "*" in place of an optimal action element in a row indicates that that action element is irrelevant in the context of the input specified in that row.

Table 5.1. *The interference task.*

Inputs $x_1\ x_2 \dots x_n$	Optimal Actions $y_1\ y_2 \cdots y_n$	Process Output z	Error e
1 0 0 … 0	0.9 * * … *	$1.0 - y_1$	$\mid 0.9 - y_1 \mid$
0 1 0 … 0	* 0.9 * … *	$1.0 - y_2$	$\mid 0.9 - y_2 \mid$
⋮	⋮	⋮	⋮
0 0 … 0 1	* * … * 0.9	$1.0 - y_n$	$\mid 0.9 - y_n \mid$
1 1 1 … 1	0.1 0.1 … 0.1	$1.0 - 1/n \sum_{j=1}^{n} y_n$	$1/n \sum_{j=1}^{n} \mid 0.1 - y_n \mid$

The task can be summarized as follows. There are two classes of vectors from which input to the controller is selected in this task. In one class there are n unit vectors, each with a single input element set to 1 and the rest set to 0. In the other class there is a single input vector with all the input elements set to 1. When a unit vector is presented as input to the controller, an action element y_i is active[4] only if the corresponding input element x_i is 1. In this case, the optimal value of the active action element is 0.9, and we do not care about the values of the other action elements. However, in the special case when all the input elements are 1, all the action elements are active, and each one of them should equal 0.1. For each input, an action error is computed using the corresponding optimal action values and taking into account only the active elements. The evaluation is then computed as a monotonically decreasing function of this error. In addition, for each input, a "process output" that is a function of only the active action elements is provided to the learning system (see Table 5.1). A typical arrangement of the controller and the process+critic for this task is shown in Figure 5.3.

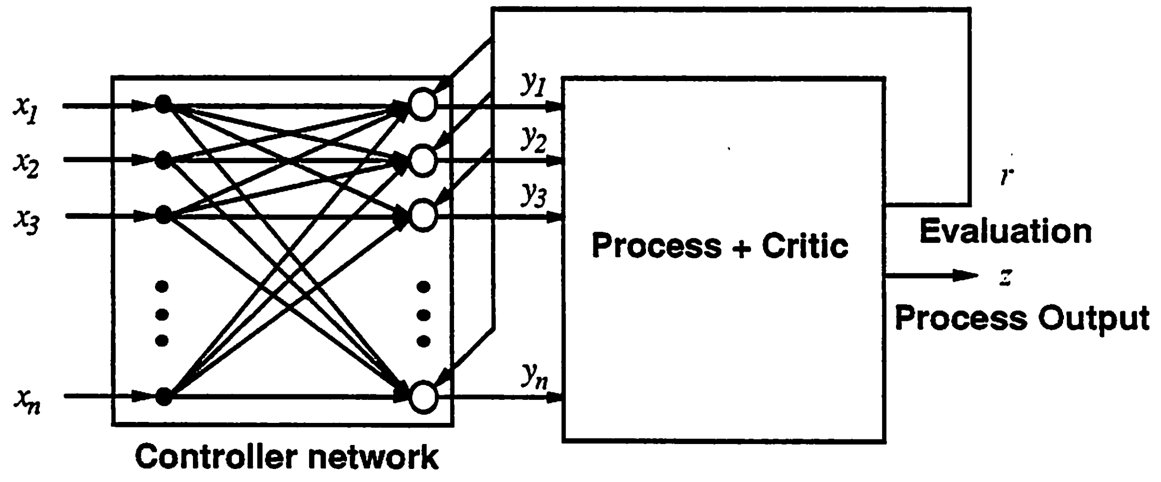


Figure 5.3. *Block diagram of the setup for the interference task. There are n input and action elements.*

Why is this task is called the "interference task?" Direct reinforcement learning involves perturbing the action elements and adjusting them based on the consequent change in the evaluation. When a unit vector input is presented to the controller in the interference task, only one action element is active and should learn to correct its error.

[4]See Section 5.4 for the definition of an active action element.

But because the evaluation signal is provided to all the action elements, the inactive action elements will also be adjusted based on the correlation between perturbations in their values and the observed change in the evaluation. This superstitious learning interferes with learning the optimal actions when the inactive action elements become active. Requiring an active action element to produce different outputs depending which input vector (a unit vector or a vector of all ones) is presented to the controller exacerbates the effect of this interference. The goal, obviously, is to minimize such interference effects through appropriate credit assignment.

The credit assignment scheme described in Section 5.4 accomplishes this by turning down the sensitivity of the inactive action elements, thereby diminishing superstitious adjustment of their values. Inactive action elements can be detected by using a local linear model (LLM) of the transformation from actions to the process output. The derivatives of the process output with respect to the control action elements will have small magnitudes (ideally, zero magnitudes) for inactive action elements and large magnitudes for the active elements. Using these derivatives to assign sensitivities to the action elements will therefore result in low sensitivities for the inactive action elements as desired.

Results

We ran simulations in which controllers with 2, 3, and 4 action elements were trained in the interference task. The results of these simulations are shown in Figures 5.4–5.6, which show the evaluation plotted as a function of time. As is evident from these figures, in all cases the system learns to solve the interference task when credit assignment among the action elements is performed but fails to solve the task when no credit assignment is performed. This is indicated by the uniformly high evaluation received in the former case. Moreover, we can observe that without credit assignment the performance becomes progressively worse as the number of action elements increases. These results suggest the utility of combining

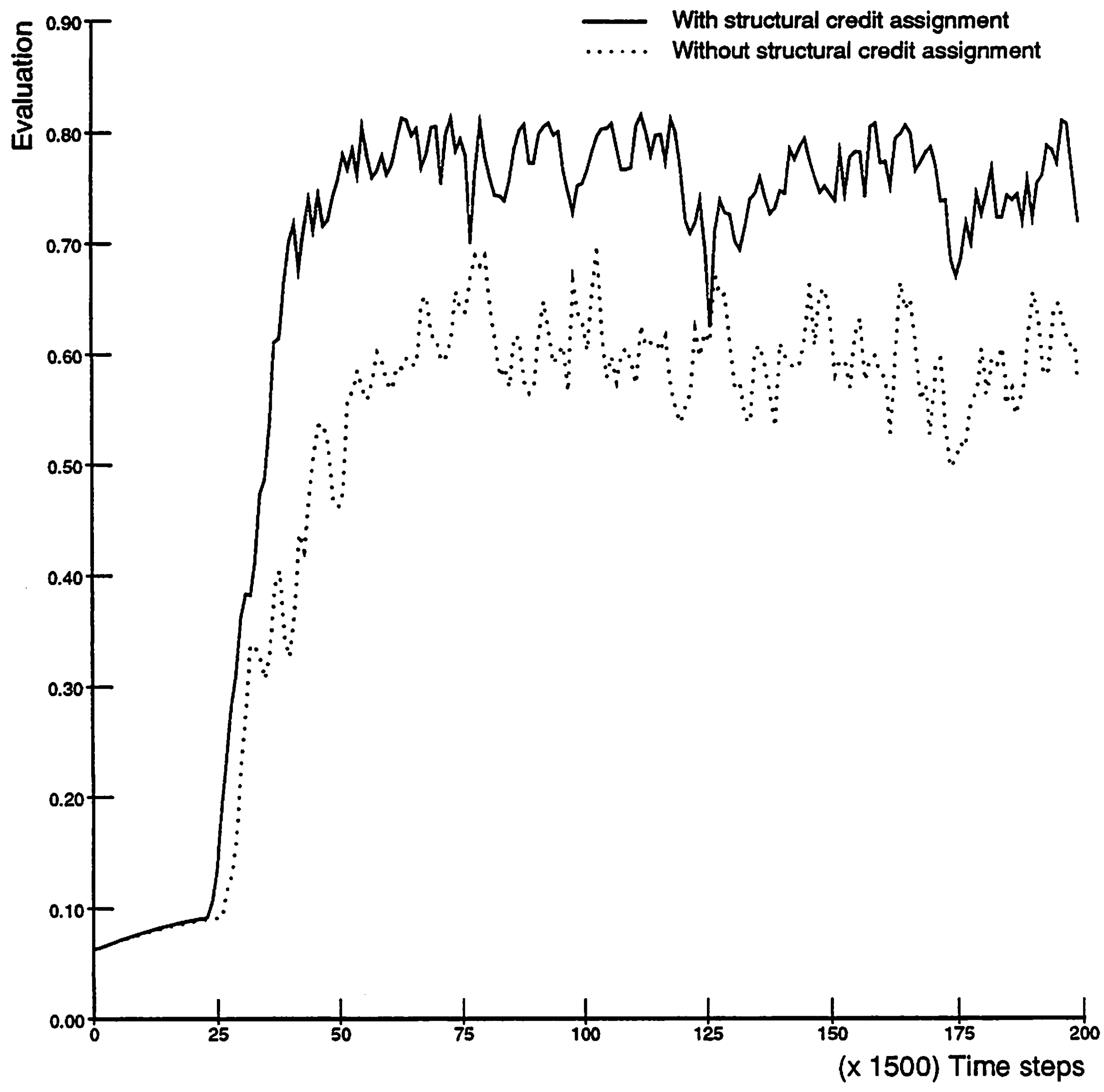


Figure 5.4. *Performance on the interference task for $n = 2$. The curves show the evaluation received at each time step, averaged over 20 training runs of 300,000 time steps.*

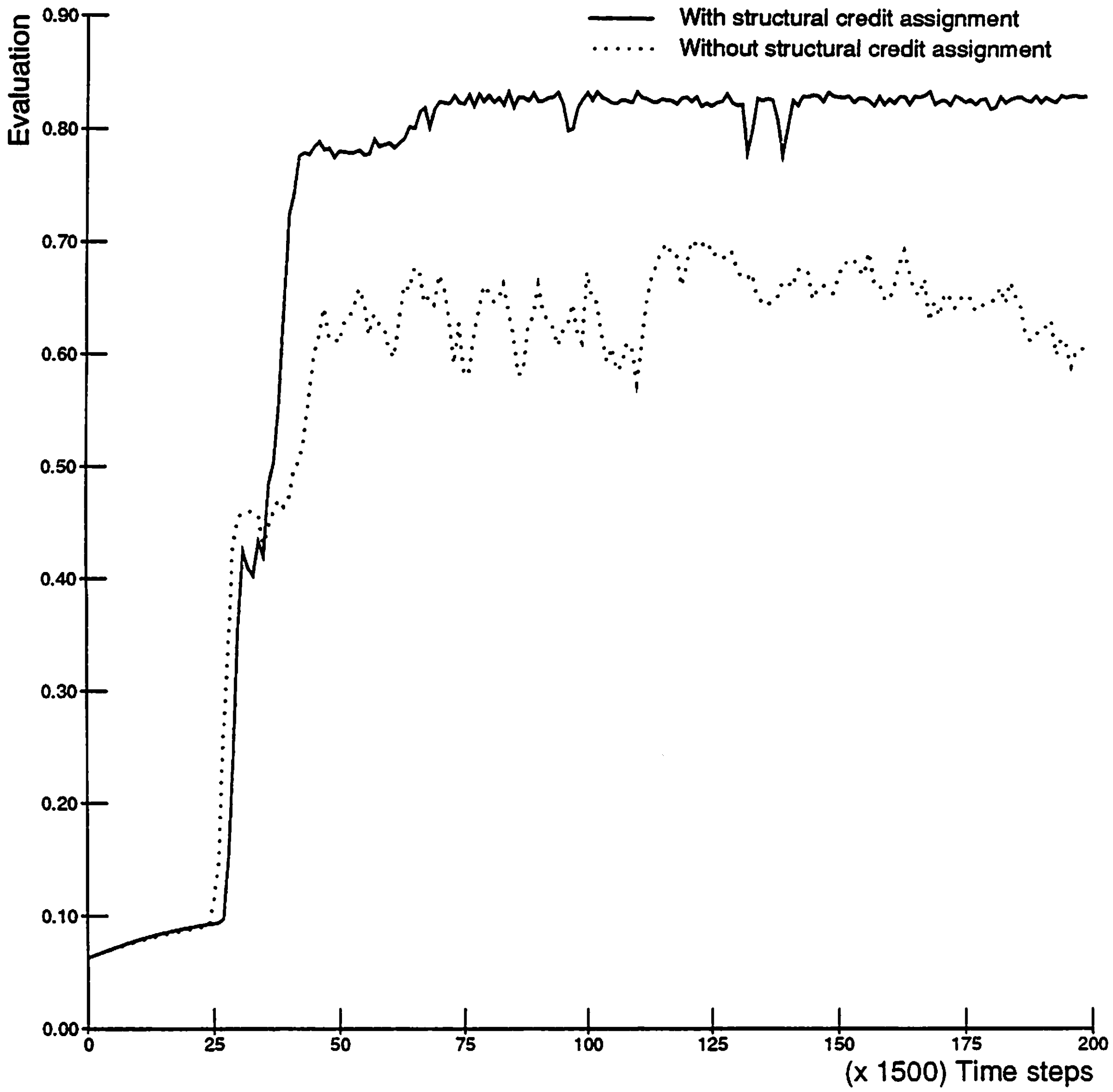


Figure 5.5. *Performance on the interference task for $n = 3$. The curves show the evaluation received at each time step, averaged over 20 training runs of 300,000 time steps.*

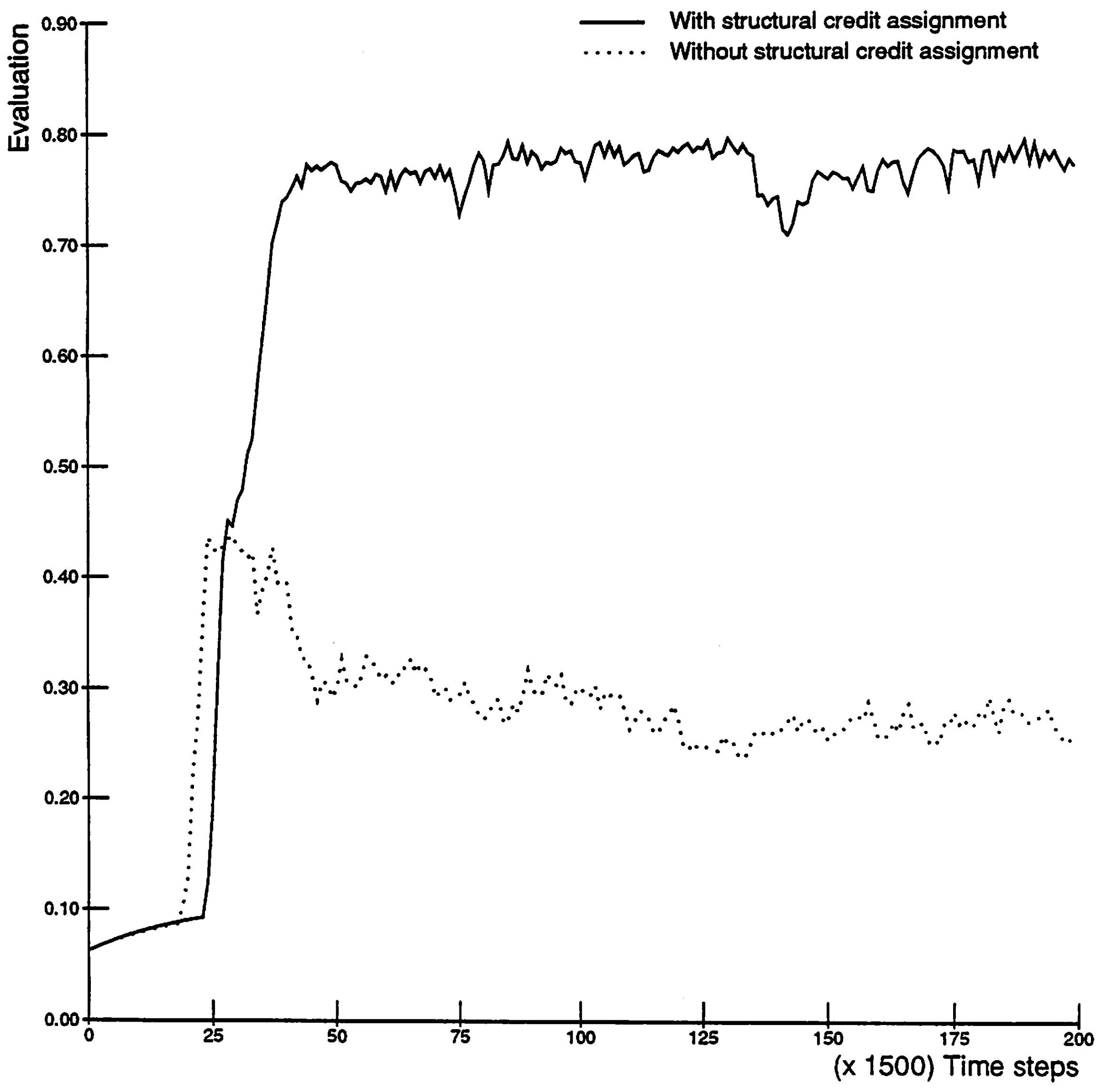


Figure 5.6. *Performance on the interference task for $n = 4$. The curves show the evaluation received at each time step, averaged over 20 training runs of 300,000 time steps.*

a multiple action element credit assignment mechanism with direct reinforcement learning methods. Although it is possible to train a controller to solve the interference task using unaugmented direct reinforcement learning methods, doing so might be untenable in practice because the learning rate has to be kept very low to ensure that the controller learns reliably despite the interference.

The results presented here also serve to illustrate how a reinforcement learning system can take advantage of the information available in the process output to improve its learning performance. Because such information is commonly available when dealing with control problems, the technique presented in this chapter is especially useful for multiple action element credit assignment in a learning controller.

Chapter 6
Shaping as a Method for Accelerating Reinforcement Learning

We begin this chapter by describing an animal training procedure used in experimental psychology called shaping. The principle underlying shaping is that learning to solve complex problems can be facilitated by first learning to solve simpler subproblems. Shaping can be implemented in two ways in a system for learning control. First, the behavior of a controller can be shaped over time by gradually increasing the complexity of the control task as the controller learns. Second, shaping can be implemented *structurally* by using a multilevel architecture trained bottom-up so that previously learned control behaviors are used as primitives for learning more complex behavior. Both these methods of shaping are illustrated here using examples.

6.1 Shaping

Shaping has been used for hundreds of years to train animals and has been studied by experimental psychologists [60] interested in animal learning. The term "shaping" itself has been attributed to the psychologist Skinner [127], who used the technique to train animals, such as rats and pigeons, to perform complicated sequences of actions for rewards. Skinner describes how the technique is used to train pigeons to peck a spot:

> We first give the bird food when it turns slightly in the direction of the spot from any part of the cage. This increases the frequency of such

> behavior. We then withhold reinforcement until a slight movement is made toward the spot. ...We continue by reinforcing positions successively closer to the spot, then by reinforcing only when the head is moved slightly forward, and finally only when the beak actually makes contact with the spot...
>
> The original probability of the response in its final form is very low; in some cases it may even be zero. ...By reinforcing a series of successive approximations, we bring a rare response to a very high probability in a short time. ...The total act of turning toward the spot from any point in the box, walking toward it, raising the head, and striking the spot may seem to be a functionally coherent unit of behavior; but it is constructed by a continual process of differential reinforcement from undifferentiated behavior, just as the sculptor shapes his figure from a lump of clay. (Skinner [129] pp. 92-93)

The phrase "...reinforcing a series of successive approximations..." expresses the essence of shaping. Given the task of training an animal to produce complex behavior, the trainer has to be able to (1) judge what constitutes an approximation to, or a component of, the target behavior, and (2) determine how to differentially reinforce successive approximations so that the animal easily learns the target behavior. Unfortunately, neither of these two components of shaping have been formalized rigorously in the psychology literature, even though shaping is widely used both in psychological studies and to train pets and circus animals. Staddon [132], for example, observes that the trainer often has to rely on an intuitive understanding of the way the animal's behavior is generated when determining which behavioral variations are precursors to the target behavior and how to reinforce these precursors. Variations in the behavior of individual animals also must be accounted for when making these judgements.

The limitations of relying on intuition when judging behavioral distances are especially obvious when the behavior under consideration is cognitive in nature (for example, learning language or mathematics). However, when the physical behavior of the animal is being shaped, behavioral distances become equivalent to physical distances, and it is therefore easier to determine a sequence of behavioral approximations that will lead to mastery of the target behavior. It is therefore not surprising that shaping has been used most often for teaching motor skills to animals. For the same reason, shaping can also prove useful for training artificial learning systems to perform as controllers.

Several connectionist researchers have noted that training a controller to perform one task can facilitate its learning a related second task (e.g., [125, 48, 151]). Selfridge, Sutton, and Barto [125] studied the effect of shaping over time when training a controller to balance a pole mounted on a cart.[1] They observed that overall learning times were typically shorter when an existing controller was retrained whenever modifications were made to the cart-pole system than when a new controller was trained from scratch. This was demonstrated for several types of modifications including increasing the mass of the pole, shortening the pole, and shortening the track.

Wieland [151] illustrated the utility of shaping using a different version of the cart-pole task in which the controller had to simultaneously balance two poles mounted on a cart. Because it is easier to solve the two-pole balancing problem when the pole lengths are very different than when the pole lengths are almost equal, Wieland trained a controller to balance poles of lengths 1.0m and 0.9m by starting with poles of lengths 1.0m and 0.1m and gradually increasing the length of the shorter pole to 0.9m. Although it is very difficult to balance poles with lengths as close as 1.0m and 0.9m, the shaping process resulted in a controller that was able to do so. Wieland and

[1]For a description of the cart-pole task, see Section 4.3.

Leighton [142] also studied the utility of shaping schedules for accelerating learning methods based on gradient descent.

Other applications of shaping in connectionist research have been in the area of training recurrent nets. Allen [4] trained recurrent nets to generate long sequences of outputs using a shaping procedure that involved initially training the nets with short target sequences and introducing longer sequences gradually over training. Another related form of shaping is described in Nowlan [109]. In this case, a robust attractor state for a recurrent network is developed by first training from initial states near the attractor, and then gradually increasing the distance of initial states from the attractor.

In this chapter, we present experimental results illustrating the utility of shaping in training controllers via direct reinforcement learning methods. There are two different ways to implement shaping. First, the behavior of a controller can be shaped over time by gradually increasing the complexity of the control task as the controller learns. This is identical to the manner in which shaping is used to train animals and to the manner in which shaping was used in the studies cited above. Second, taking advantage of the ability to specify the structure of an artificial learning system, one can implement shaping *structurally* by using a multilevel architecture for the controller. The idea here is to train the multilevel controller bottom-up so that previously learned control behaviors are used as primitives for learning more complex behaviors. In the following sections, we examine both of these approaches to shaping a controller's behavior in greater detail .

6.2 Shaping through differential reinforcement of behavior over time

In this section, we present an example illustrating how a reinforcement learning controller's behavior can be shaped over time to produce a complex behavior. As described above, in order to shape the behavior of the controller, one has to determine

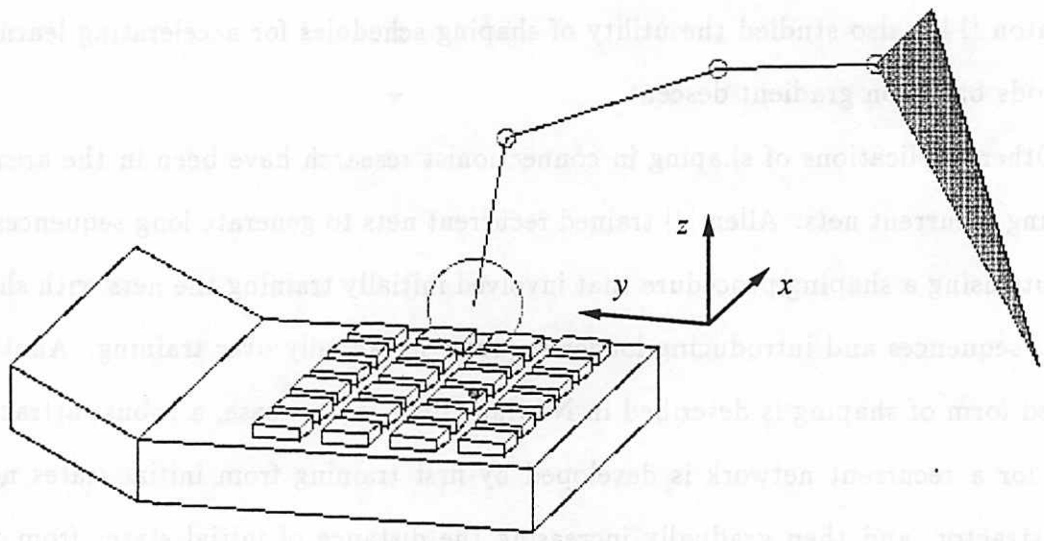


Figure 6.1. *The task setup for the key-pressing task. The simulated Stanford/JPL hand and the calculator are shown to scale. Only the index finger of the hand is shown because only that finger is used in the task. The triangle represents the palm of the hand; the large circle represents the fingertip. The small dark circle on the calculator face is the "footprint" of the center of the fingertip.*

(1) a series of approximations to the target behavior and (2) how to differentially reinforce successive approximations to the target behavior.

6.2.1 A Test Task: Key-pressing using a robot hand

We demonstrate the utility of shaping over time using a control task that involves pressing keys on a simulated calculator keypad using the index finger of a simulated dynamic model of the Stanford/JPL hand. The finger has three degrees of freedom and the motion of the hand is restricted to a plane parallel to the x-y plane in which the calculator face lies. Thus there are five degrees of freedom in all to be controlled. The task setup is depicted in Figure 6.1. The axes of rotation of the finger joints of the Stanford/JPL hand are as follows: the first joint (linking the finger to the palm) permits rotation about an axis parallel to the z-axis, the other two joints have axes of rotation that are perpendicular to both the first link of the finger and the z-axis.

The control actions are positioning commands that locate the hand base in its plane of motion and position the three joints of the index finger. The key to press is specified by setting a single bit in a 24 bit command input vector supplied to the controller. Additional inputs to the controller include proprioceptive feedback of the positions and velocities of the finger joints and the hand, a fingertip force sensation, and a binary "key-pressed" sensation that is set whenever a key is pressed and reset whenever a new target key is specified. For successfully pressing a key, the fingertip must depress the key to the level of the face of the calculator ($z = 0$). Provision of the hand position and velocity feedback permits the controller to learn to press any key starting from any initial hand configuration.

Because pressing a key involves positioning the fingertip over the key, pressing it, and then releasing, the task is fairly complicated and the controller has to learn a sequence of actions for executing a single key-press operation. As one might imagine, the probability of a reinforcement learning controller generating such a complex behavior through stochastic search is infinitesimal. A simple evaluation criterion that only signals successful key-presses is therefore not very useful for training the controller. Fortunately, we can shape the control behavior by defining successive approximations to the key pressing operation and providing more informative differential evaluations that can facilitate learning.

The first problem in implementing shaping, which is to determine a series of approximations to the target behavior, is not very difficult for the key-pressing task. A fairly intuitive series of approximations to the key-press operation is the following:
(1) Raising the fingertip so that it is not in contact with the keypad surface (to prevent accidental key strikes).
(2) Moving the fingertip towards the target key keeping the fingertip raised.
(3) Positioning the fingertip over the target key keeping the fingertip raised.
(4) Positioning the raised fingertip over the target key and then pressing down with the fingertip.
(5) Positioning the raised fingertip over the target key and pressing down until the key is fully depressed.

(6) Positioning the raised fingertip over the target key, pressing down until the key is fully depressed, and then releasing the key by raising the fingertip.

The second problem in implementing shaping, i.e., deciding how to differentially reinforce control behavior, is more complicated. In order to differentially reinforce the controller as it learns a series of approximations, the critic has to maintain a behavioral history of the controller and infer from the history how well the controller has learned each approximation to the target behavior. Based on this inference, the critic has to determine if the controller requires further training on a particular approximation or if it is ready to be trained on the next, more sophisticated, approximation. While it is wasteful to continue training the controller on an approximation that it has already mastered, switching to a more sophisticated approximation too quickly can also be detrimental to rapid learning of the target behavior. To best realize the benefits of shaping, accurate judgement of the controller's ability at every stage in training is therefore necessary. Clearly, the time frame over which the behavioral history is maintained and evaluated is a factor in making these judgements.

Our approach to the problem of differentially reinforcing the controller's behavior is to (1) reduce the time frame over which the controller's behavior is evaluated to individual training runs, i.e., individual attempts at the target behavior, and (2) require that the controller's behavior progressively satisfies the criteria for all the approximations to the target behavior, starting with the simplest, *in each attempt* at the target behavior. This approach allows the controller's goal to be switched to increasingly sophisticated approximations quickly over training, while at the same time ensuring that the controller is trained on an approximation only if it has successfully met the criteria for all simpler approximations. Moreover, this approach sidesteps the question of how to infer the controller's ability at a stage in training from its behavioral history over a longer time frame.

In the key-pressing task, for example, the critic maintains the behavioral history of the controller only over individual training runs, which begin with a new target key being assigned and end after a fixed number of time steps have elapsed. During the course of each training run, the performance of the controller is evaluated using a series of criteria, each attuned to a corresponding approximation in the list above. The criterion used to determine the evaluation is selected at each time step based on the state of the hand and the portion of the above series of approximations that has already been accomplished. For example, if the fingertip is raised but is not located over the target key, an evaluation criterion that rewards motion towards the target key keeping the fingertip raised is selected; if the fingertip has already been positioned over the target key by the controller, the criterion selected rewards downward motion of the fingertip while keeping it located over the target key; and so on.

Initially, the controller might spend the entire duration of a training run learning to satisfy the criterion for the simplest approximation. With time, the controller learns to consistently satisfy the criteria for the simpler approximations, and the frontier of learning shifts to approximations closer to the target complex behavior. Thus, most of a training run is spent in learning the approximation to the target behavior at the current frontier of learning.

6.2.2 Training methodology

In order to keep computer simulation time reasonable while retaining all the essential aspects of the key-pressing task, we restricted the choice of the target keys to three keys, which are highlighted in Figure 6.2. These keys were chosen so as to require a broad range of motion of the fingertip. The controller was trained in a series of training runs, which began with a new target key being picked randomly. The probability of picking the key used in the previous training run was 0.1, while that of picking either of the other two keys was 0.45. The initial hand configuration used in a training run was the configuration of the hand at the end of the previous training run. However, if the previous training run left the fingertip touching the keypad, the last two finger joints were repositioned so that the fingertip was no longer in contact with

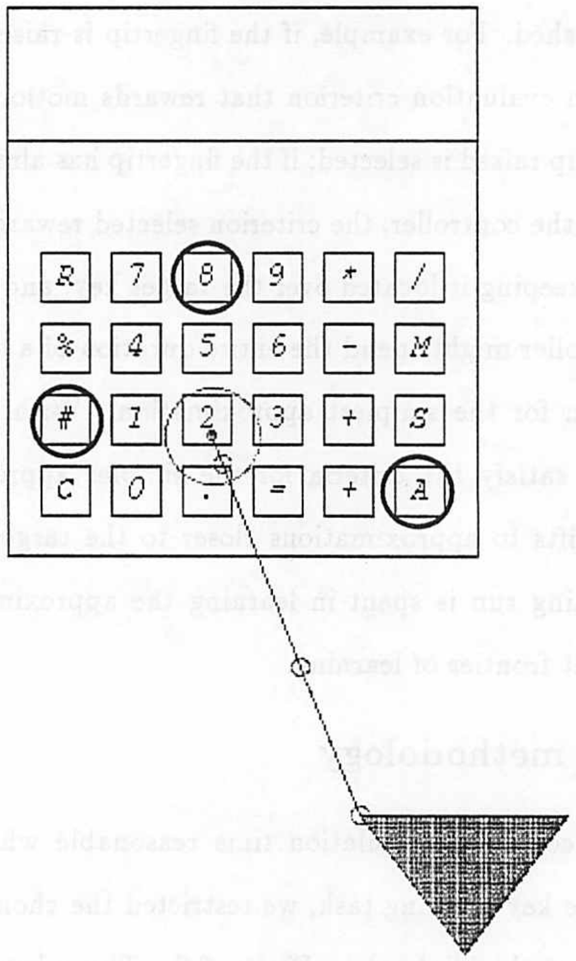


Figure 6.2. *The set of target keys used in the key-pressing task, shown encircled by bold circles.*

the keypad. In either case, the initial velocities were set to zero. Each training run lasted 15 time steps, during which the controller was trained using the appropriate evaluation criterion at each time step as described above. The sensory feedback to the controller was also updated at each time step. Details of the implementation of the controller and the simulation of the calculator/hand system are given in Appendix B.

For the purpose of comparison, we also attempted to train a controller on the key-pressing task without resorting to shaping. An identical training procedure was followed in this case, with the sole modification being the use of a single evaluation criterion that only rewarded pressing of the target key.

6.2.3 Results

The performance of the controller on the key-press task after 25,000 training runs is shown in Figures 6.3–6.5. Each figure contains four panels that show the motion of the hand over twelve time-steps when pressing each of the three target keys. Panel (a) contains three strip-charts that show the value of each of three quantities at each time step over the course of a key-press operation. These are the distance of the fingertip to the target key (marked D and ranging from 0 to 9 centimeters), the height of the fingertip above the calculator face (marked Z and ranging from 0 to 1 centimeter), and the payoff, or evaluation, (marked P and ranging from 0 to 1). Panels (b), (c), and (d) show the motion of the hand during the key-press operation from three different viewpoints.

The strip-charts show that the fingertip is lowered as it approaches the target key until it makes contact with the key and begins to depress it, and, once the key is fully depressed, the fingertip is raised to release the key. The sharp drop in the evaluation on the time step when the key is fully depressed (i.e., to $z = 0$) is due to the switching of the evaluation criteria from one that rewards downward movement to one that rewards upward movement.

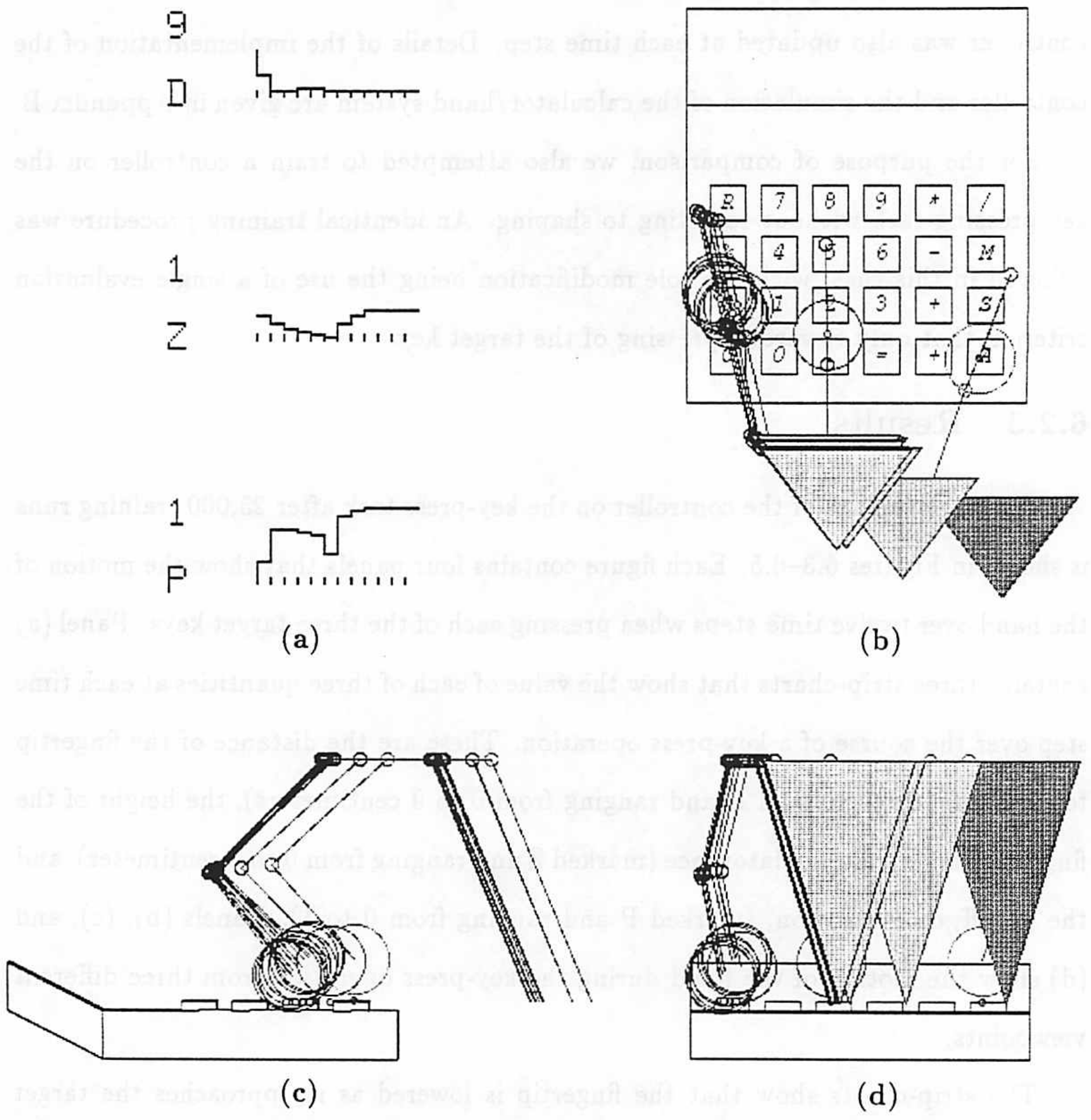


Figure 6.3. *Pressing the key marked "#" on the calculator keypad. The initial position of the fingertip was the final position after pressing the key marked "A". The dark triangle denotes the initial location of the hand.*

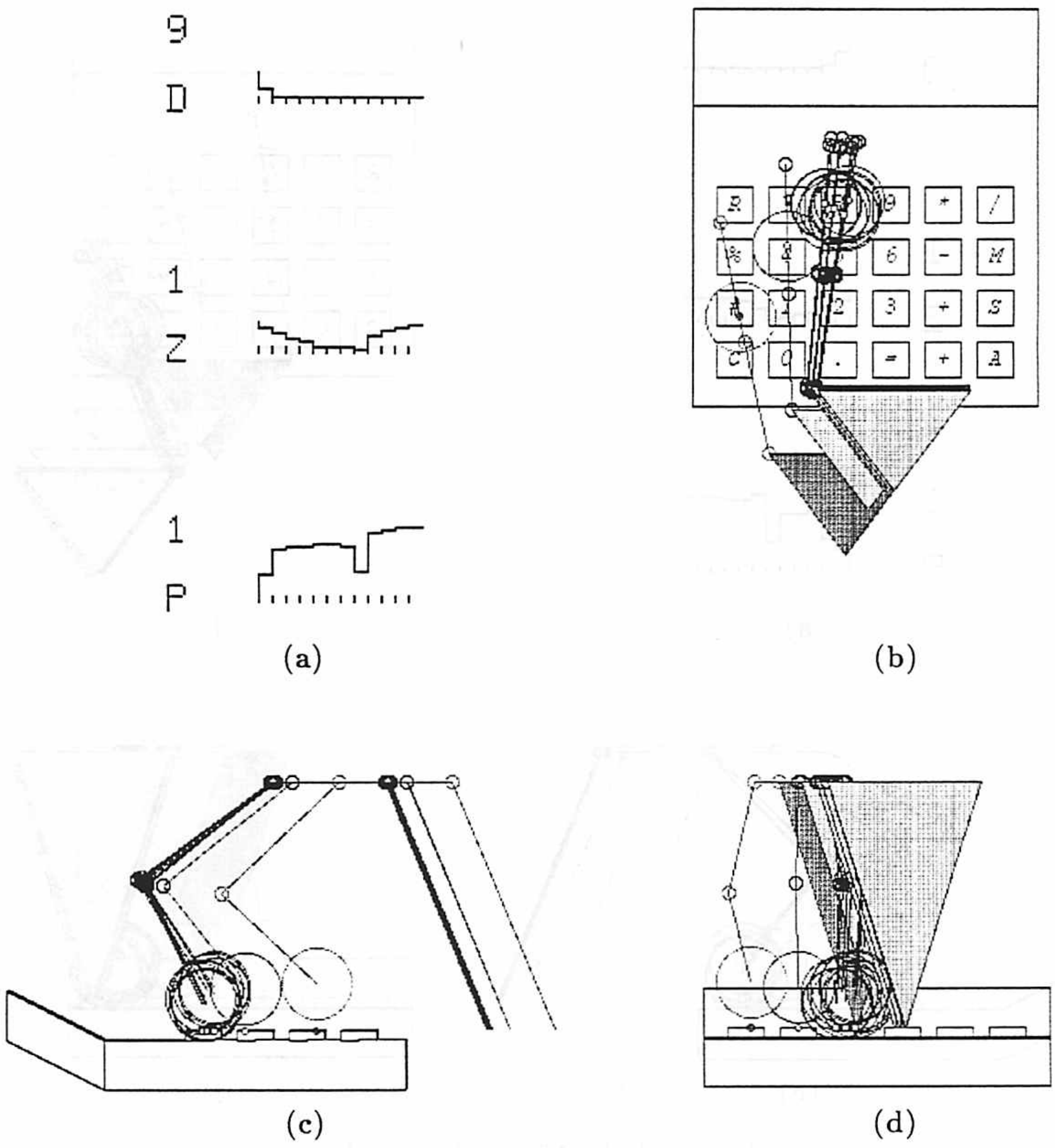
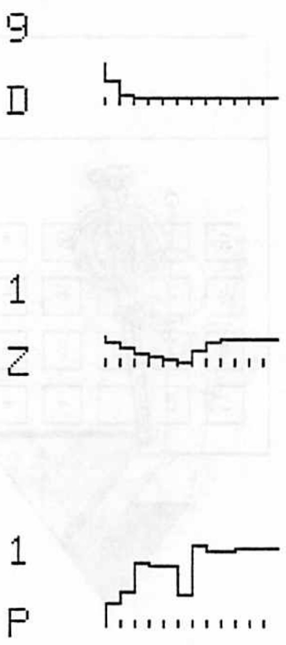
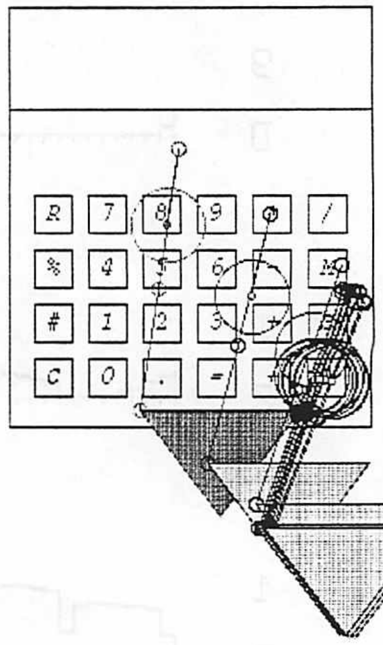


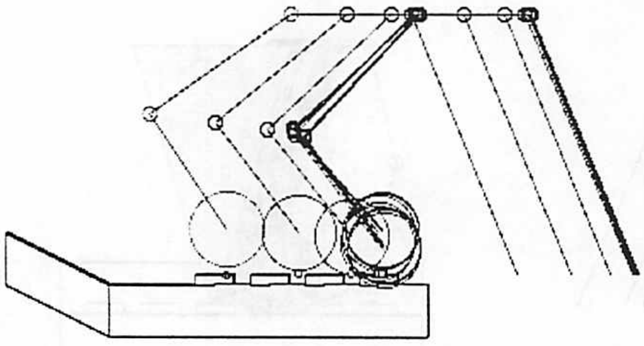
Figure 6.4. *Pressing the key marked "8" on the calculator keypad. The initial position of the fingertip was the final position after pressing the key marked "#". The dark triangle denotes the initial location of the hand.*



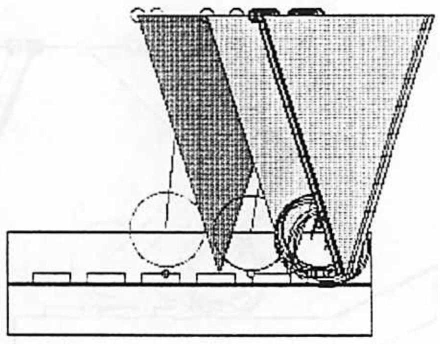
(a)



(b)



(c)



(d)

Figure 6.5. *Pressing the key marked "A" on the calculator keypad. The initial position of the fingertip was the final position after pressing the key marked "8". The dark triangle denotes the initial location of the hand.*

As evidenced by these figures, the controller has learned to successfully execute the key-press operation for all three keys. Note that due to the dynamic nature of the hand model, the motion of the fingertip depends on the initial state of the hand. So the figures presented here are merely representative samples. However, we tested the controller's performance with the hand starting in 10,000 random initial states, and in all the test runs, the target key was pressed successfully.

In comparison, without shaping, the controller could not learn the key-press task even after 500,000 training runs. Moreover, this was true even when the controller was trained to press just a single target key (the key marked "8"). These results support the observations in Section 6.2.1 regarding the difficulty of training a reinforcement learning controller to generate complex behavior without the benefit of shaping due to the improbability of occurrence of the target complex behavior.

6.3 Shaping through incremental development of the learning system

In this section, we explain, using an example, how a reinforcement learning controller's behavior can be shaped by incrementally augmenting the structure of the controller and training it on increasingly complex tasks. As the key-pressing task of the previous section illustrates, for most complex tasks, the controller has to translate the specified task into a sequence of control actions. In such cases, it might be advantageous to decompose a complex high-level task into a sequence of subtasks of a lesser degree of difficulty [137, 25]. This process of decomposition can be repeated for subtasks at each succeeding level until a subtask is decomposed into a sequence of elemental control actions. Such a decomposition enables us to represent any task as a trajectory of subtasks at each level of decomposition (see Albus [3]).

A set of tasks is defined to be *decomposable* if the set can be divided into at least two disjoint subsets of tasks of different levels of difficulty, such that (1) any task of a level of difficulty can be decomposed into a sequence of tasks of lower levels of difficulty, and (2) the tasks at the lowest level of difficulty can be directly decomposed into a sequence of control actions. The set of disjoint subsets is called the *decomposition set* of the set of tasks. If a controller is to perform a decomposable set of tasks, we can take advantage of the decomposability by designing a controller that is split into several levels, with each level performing the decomposition of the subtasks at that level.

Such a controller architecture has several advantages. First, the decomposition of tasks into subtasks simplifies the control problem because the controller can try to solve each subtask independently. Since the subtasks are simpler than the original task, such a divide-and-conquer approach decreases the complexity of the learning process. Moreover, the control techniques and procedures required for solving the subtasks are less sophisticated than those required for solving the whole task, leading to a much simpler overall design for the controller.

A third advantage of a multilevel architecture for the controller is the ability to develop the controller in a modular fashion, one level at a time. We can start with a single-level controller that can perform simple tasks and incrementally add higher levels that enhance the power of the controller. In other words, we can shape the controller's behavior through incremental development of its structure. This shaping procedure also permits us to ensure that the performance of the controller at each stage meets prespecified standards. Another advantage of training a multilevel architecture to perform task decomposition is that it permits sharing of control knowledge between tasks when they share a common set of subtasks.

Finally, the decomposition of tasks and assignment of subtasks to each level offer excellent opportunities for the incorporation of domain knowledge into the design of the controller. The usefulness of domain knowledge in problem-solving has been stressed by several AI researchers (for example, Minsky [96] and Newell and Simon [106]). Because each control problem has its own requirements and characteristics that may, or may not, be easily handled by general-purpose control methods, the ability to incorporate domain knowledge into the design of the controller adds a useful degree of flexibility.

Multilevel and hierarchical controllers have been proposed by Albus [2], Minsky [94], Saridis [121], Meystel [91], and others, as effective mechanisms for tackling difficult control problems. These researchers have been inspired by the ubiquity of multilevel systems in the brain, and in ecological, social, and economic systems. Although these researchers have described various multilevel and hierarchical architectures for control, very few have actually addressed the problems of learning in such architectures. Examples of learning in a hierarchical organization of modules have been presented by Ersü and Tolle [36], Miyata [97], Jacobs [63], and Nowlan and Hinton [110].

Although the task decomposition perspective presented above is useful, the idea of shaping through bottom-up development of the controller is perhaps more closely related to the ideas of learning *macro operators* [77, 61] or *chunking* [79] from the problem solving literature of Artificial Intelligence. Korf [77] proposed learning macro operators, or fixed sequences of actions that accomplish various subtasks, as a means of speeding up learning to perform a complex task. In a similar vein, Laird, Newell, and Rosenbloom [79] proposed chunking as a mechanism (incorporated into a planning and problem solving system called SOAR) for storing generalized versions of previously generated plans. Learning macro operators or chunking can lead to considerable speed-up when learning to perform a complex task by eliminating repeated search for fixed action sequences that are used frequently.

Similar mechanisms for storing and using precompiled plans or precompiled sequences of actions have been suggested by several other AI researchers. The use of MACROPs to save generalized versions of plans previously generated by the robot planning system, STRIPS, has been examined by Fikes, Hart, and Nilsson [39]. PULP-I, designed by Tangwongsan and Fu [139], is another example of a robot planning system that uses stored skeletal plans to speed up the planning process.

The effective use of stored skeletal plans also helps PULP-I reduce the number of operators used in a plan for a task, leading to more efficient plans.

Bottom-up development of a multilevel controller is analogous to learning and using macro operators. To see this analogy, consider incrementally developing a two-level controller by training the lower level first and then adding and training the higher level. In the case when there are more than two levels, this analogy holds for each pair of consecutive levels. Training the lower level of a two-level controller on a subtask is similar to learning a macro and involves searching for the right sequence of actions to accomplish the subtask. The "name" of the subtask, i.e., the corresponding command input to the lower level, serves as the macro label. By associating the appropriate sequence of actions with each macro label, the lower level essentially functions as a macro interpreter. When the higher level is added to the control architecture, it has to learn to generate an appropriate sequence of commands to the lower level in order to perform the higher level task. Moreover, because the only valid commands to the lower level are the macro labels that it can interpret, the command sequence from the higher level should be composed of only these macro labels.

This last observation is of importance when implementing a two-level (or, in general, multilevel) controller. Depending on the representation used for the command input to the lower level in an implementation, the number of valid commands could be far less than the number of possible inputs using that representation. For example, if an n-bit binary vector is used to represent the commands to the lower level, and the lower level has been trained on m different macros, the actual size of the search space for the higher level's actions is only m even though there are 2^n n-bit binary vectors. Often, m is considerably less than 2^n, and hence the learning problem for the higher level can be reduced considerably by restricting its search for appropriate action sequences to the set of m valid commands.

However, when using direct reinforcement learning methods, it becomes difficult to restrict the search to only valid commands because randomly perturbing the command output generated by the higher level might not result in a valid command to the lower level. An alternative approach, therefore, is to ensure that the lower level gets only valid command inputs even though the higher level does not always generate valid commands. One way of accomplishing this is to divide the space of all commands representable using a given command representation into equivalence classes, one for each valid command, and to learn a mapping from an arbitrary command to the valid command associated with its equivalence class. Once such a mapping has been learned, it can be used to transform any command from the higher level into a valid command for the lower level.

In order to implement this idea, a special module, which we call the *command filter*, can be trained to perform the mapping defined above in parallel with training the lower level. By filtering all commands from the higher level—when one has been added to the control architecture—through the command filter, we can ensure that the lower level always has valid commands to execute. Autoassociative memory networks (e.g., [76, 57]) are examples of connectionist networks that can learn to perform the kind of mapping described above and hence can be used to implement the command filter modules. In the next section, we present an example demonstrating the advantage of employing a command filter module implemented using an autoassociative connectionist network.

But before we do that, we briefly consider the issue of timing between two consecutive levels in a multilevel controller. Because the lower level produces a sequence of actions for each command it receives from the higher level, and the higher level issues a sequence of commands to the lower level, there is a problem of synchronization between the two levels. The operation of the two levels has to be coordinated so that each command issued by the higher level is executed completely by the lower level.

There are several approaches to enforcing proper coordination between the two levels, although their practicality is still an open question. For example, if the execution of any command always takes less than a fixed duration, the higher level could wait for that duration before generating the next command. Or the lower level could always execute each command completely before accepting the next command from the higher level, ignoring any commands issued while it is busy. This begs the question of how to ensure that all commands issued by the higher level are executed. One approach is for the lower level to maintain a stack of commands received from the higher level while it was busy rather than ignoring them. Alternatively, after it has finished executing the current command, the lower level could send a special signal triggering the higher level to generate the next command. In this case, the higher level of the controller could hold a command fixed until the lower level has completed executing it and then issue the next command. Finally, instead of relying on the lower level to provide a trigger signal, the higher level can use sensory feedback from the environment to detect when the lower level has finished executing the current command.

With the help of an example, we now illustrate the utility of structural shaping through bottom-up development of a multilevel control architecture.

6.3.1 A Test Task: Learning to count

The task selected to illustrate the utility of a multilevel control architecture is a bead counting task adapted from Oliver Selfridge's "Learning to count. How a computer might do it" [124]. This task requires learning sequences of actions for counting using an abacus. The task setup, shown in Figure 6.6, consists of a string of beads, a pointer, and a counter. The pointer is used to separate the beads into two groups, one on its either side. The counter can hold an integer between 0 and 99. The four primitive operations are: move the pointer Left (L) or Right (R) one bead, and Increment (I) or Decrement (D) the counter by 1. Although more than one operation can be attempted at the same time, contradictory combinations and impossible operations have no effect. Thus, LR or ID together, D when the counter

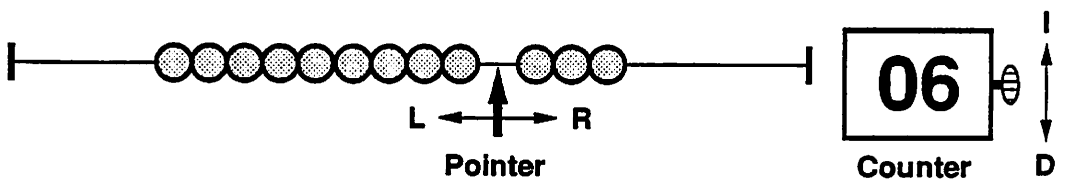


Figure 6.6. *The setup for the bead counting task.*

is 0 or I when it is 99, and L (R) when there are no beads to the left (right) of the pointer, all have no effect on the state of the counter or the pointer.

In each instantiation of the abacus, it is initialized with some number of beads, the pointer is placed at a random location along the string of beads, and the counter is set to a number between 0 and 99. The task is to COUNT, i.e., to set the counter to the number of beads on the abacus.

Learning to count the number of beads in a fixed instantiation of the abacus is straightforward, and a sequence of primitive operations that achieves this can be learned fairly easily.[2] The more interesting problem is learning to count the number of beads in *any* instantiation of the abacus. Two approaches to this problem are (1) training a single-level controller to count by generating a sequence of primitive actions, and (2) training a multilevel controller to count by first learning useful subtasks and then learning to use these subtasks to count. Our attempts to implement both these approaches are described below.

Implementation of the first approach above is fairly straightforward. However, the second approach involves defining an appropriate decomposition set of tasks and subtasks and training a multilevel controller on the tasks in this set. For the bead counting tasks, we defined the task decomposition set to contain two levels of tasks. The eight level 1 tasks are:

1. LE: Move the pointer to the left end of the bead string.

[2]Any sequence of operations that increments or decrements the counter an appropriate number of times based on the initial counter value and the number of beads is a solution.

2. RE: Move the pointer to the right end of the bead string.

3. ZERO: Decrement the counter to zero.

4. LD: Move the pointer to the left by one bead AND decrement the counter by 1. If there are no more beads to the left of the pointer or the counter is zero, do nothing.

5. LI: Move the pointer to the left by one bead AND increment the counter by 1. If there are no more beads to the left of the pointer or the counter is 99, do nothing.

6. RD: Move the pointer to the right by one bead AND decrement the counter by 1. If there are no more beads to the right of the pointer or the counter is zero, do nothing.

7. RI: Move the pointer to the right by one bead AND increment the counter by 1. If there are no more beads to the right of the pointer or the counter is 99, do nothing.

8. NOOP: Do nothing.

The single level 2 task is to COUNT.

The multilevel controller was shaped by first training a single level on all the level 1 tasks. Once the first level of the controller could perform all its tasks well, a second level was added to the controller architecture. The second level was trained to COUNT by generating sequences of level 1 tasks, which were supplied as commands to the first level. The first level of the controller, in turn, produced the appropriate sequence of primitive actions for each level 1 task command it received.

In our implementation, the problem of timing between the two levels was resolved by requiring that the second level hold a command fixed for the duration of its execution by the first level. The second level had to determine when the first level had completed execution of a command from the environmental sensory feedback (described below).

The evaluations for all level 1 tasks were binary-valued, with the evaluation becoming 1 only when the task was performed successfully. For example, if the task was LE, the evaluation was 0 until the pointer was moved to the left end of the bead string, upon which it became 1. Once the evaluation became 1, it remained 1 as long as the controller generated NOOPs.

For the COUNT task, two subtasks were defined for the purposes of computing the evaluation: moving the pointer to either end of the bead string, and decrementing the counter down to zero. These could be accomplished in any order as long as the subgoal accomplished earlier was not undone while accomplishing the second subgoal. The evaluation for the COUNT task was non-zero only when either subgoal was accomplished *for the first time*, at which time it was set to 0.9, and when the counter was set to the number of beads *after* both subgoals have been accomplished, at which time it was set to 1.0. This same evaluation was used for training the single-level controller to count.

Eight binary valued sensations were used to provide information about the state of the abacus to the level 1 controller. These are:

1. L-END = 1 if pointer is at the left end of the string.

2. R-END = 1 if pointer is at the right end of the string.

3. P-MID = 1 if neither L-END nor R-END is 1.

4. P-MOVED = 1 if the pointer changed in the last time step.

5. C-ZERO = 1 if the counter is zero.

6. C-MAX = 1 if the counter is 99.

7. C-MID = 1 if neither C-ZERO nor C-MAX is 1.

8. C-MOVED = 1 if the counter changed in the last time step.

The default value for all these sensations was 0. For the level 2 controller, three additional binary sensations were used. These are:

1. Z-REACHED. Changes from 0 to 1 when the counter becomes 0 for the first time in a training run.

2. L-REACHED/R-REACHED. Changes from 0 to 1 when the pointer is moved to the left (right) end of the bead string for the first time in a training run.

All eleven sensations were also provided to the single-level controller.

6.3.2 Training Methodology

As mentioned above, the multilevel controller was trained in two stages. In the first stage, a single level was trained on the level 1 tasks until it could perform all the tasks well. Then, in the second stage, another level was added to the controller architecture, and this second level was trained to COUNT by generating sequences of level 1 tasks, which were supplied as commands to the first level. The first level of the controller, in turn, produced the appropriate sequence of primitive actions for each level 1 task command it received.

Level 1 The level 1 tasks were specified by providing 8-bit long unit vectors as command inputs to the level 1 controller. As in the Stanford/JPL hand control example (Section 6.2.1), training runs of fixed duration (50 time steps) were used to train the controller. At the start of each training run, one of the eight level 1 tasks was selected at random, and a random instantiation of the abacus with 1 to 3 beads and a number between 0 and 3 on the counter was created. The sensory inputs to the controller were then set based on the initial configuration of the abacus, and training began. Each training run lasted for 50 time steps or until the controller received an evaluation of 1 for four consecutive time steps. The sensory inputs were updated at each time step.

The level 1 controller's actions were encoded as a 4-bit vector, with each bit denoting one of the four primitive actions (L, R, I, and D). Setting any bit to 1 at a time step invoked the corresponding primitive action at that time step. This action representation permitted the controller to manipulate both the pointer and the counter in a single time step. However, as mentioned earlier, attempting contradictory or impossible action combinations resulted in no change in the pointer and the counter.

Level 2 The level 2 controller was always trained on the COUNT task. A training run started with a random instantiation of the abacus with 2 to 4 beads and a number between 0 and 5 on the counter and lasted for 30 time steps or until the evaluation received was 1. The sensory feedback to both levels was updated at each time step of the training run.

Note that in order to generate meaningful task commands for the first level, the second level of the controller had to generate 8-bit unit vectors. Because the second level was using the direct reinforcement learning approach of perturbing each of the 8 bits randomly in order to determine the best command to output to the first level, it was performing a biased random search in a space of $2^8 = 256$ elements, even though only eight of those elements were valid task commands for the first level. Therefore this was a good example to test the utility of the command filter module in a multilevel control architecture. Simulation results from training the controller with and without the command filter are reported in the next section.

6.3.3 Results

Level 1 The level 1 controller could learn all eight level 1 tasks in 5000 training runs. Samples of the controller's performance in these tasks are shown in Figures 6.7-6.14. Instantiations of the abacus used in these tests were generated randomly by picking 1 to 20 beads and by setting the counter to a random number between 0 and 10. For each task, the figures show the initial and final abacus configurations.

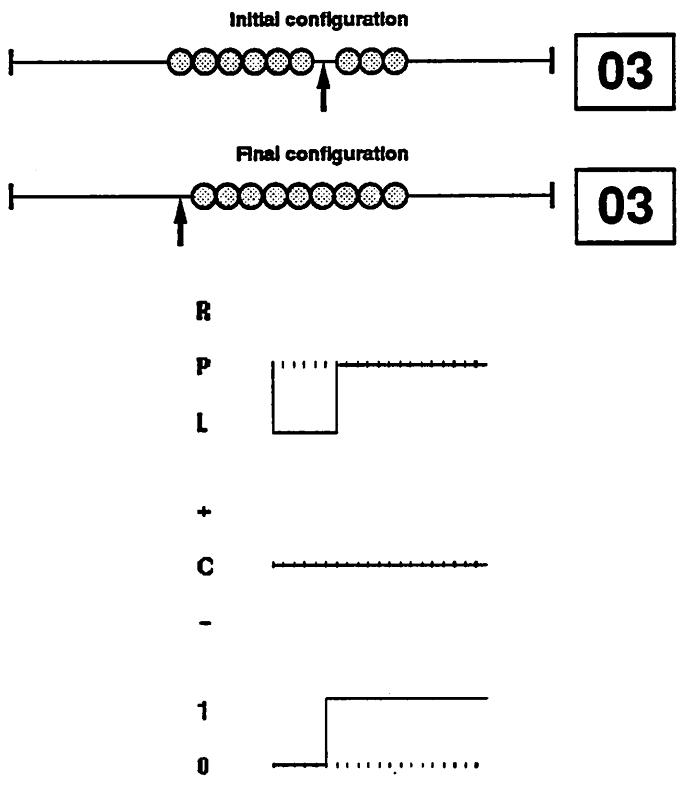


Figure 6.7. *Performance of the level 1 controller on the LE (Left End) task.*

In addition, strip-charts showing the sequence of actions produced by the level 1 controller for executing the task are also presented. The first and second rows of these strip-charts show how the pointer and the counter respectively were manipulated at each time step during the execution of the task, and the bottom row shows the evaluation received at each time step.

Note how in the case of the LI, LD, RI, and RD tasks, the controller has learned to stop (i.e., output NOOPs) whenever a limit, such as the end of the bead string or the minimum value of the counter, is reached. This is in keeping with the definitions of these tasks.

Level 2 In our simulations, when a single-level controller was trained to generate a sequence of primitive actions to perform the COUNT task, it failed to learn to count even after 1,000,000 training runs. Moreover, the use of the command filter module was also crucial in training the multilevel controller. Without the command filter,

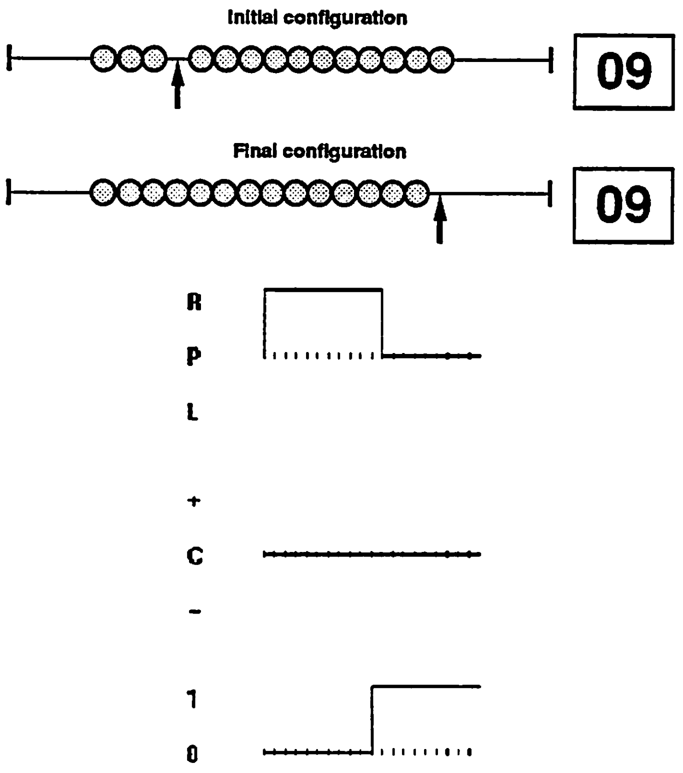


Figure 6.8. *Performance of the level 1 controller on the RE (Right End) task.*

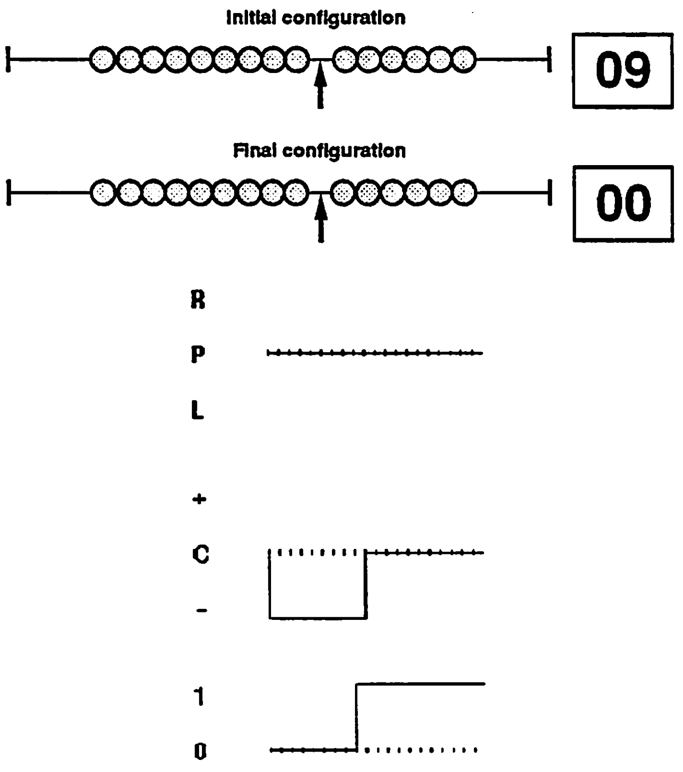


Figure 6.9. *Performance of the level 1 controller on the ZERO task.*

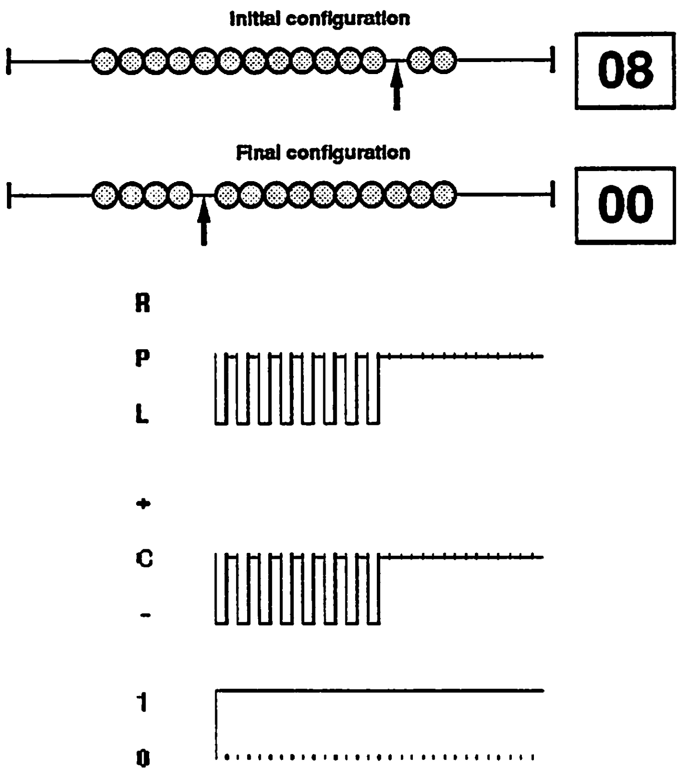


Figure 6.10. *Performance of the level 1 controller on the LD (Left Decrement) task.*

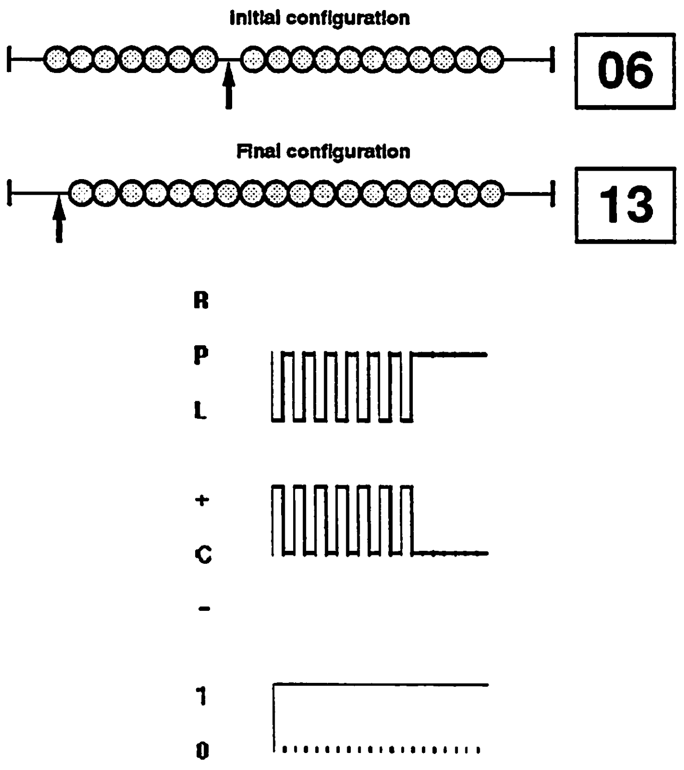


Figure 6.11. *Performance of the level 1 controller on the LI (Left Increment) task.*

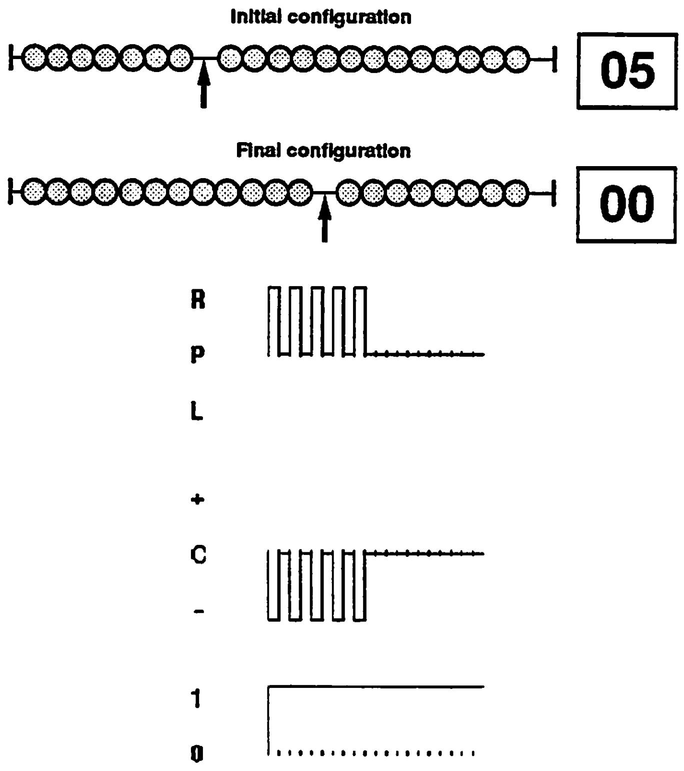


Figure 6.12. *Performance of the level 1 controller on the RD (Right Decrement) task.*

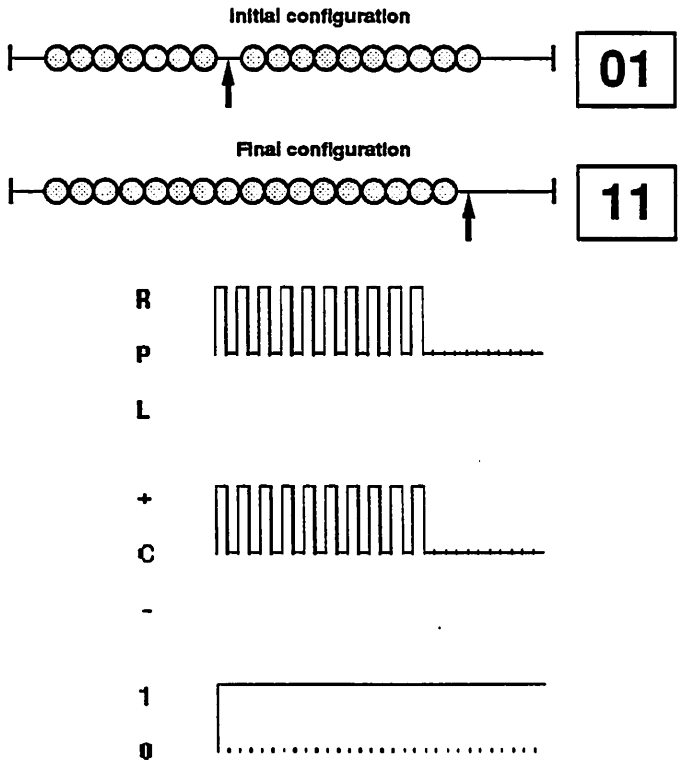


Figure 6.13. *Performance of the level 1 controller on the RI (Right Increment) task.*

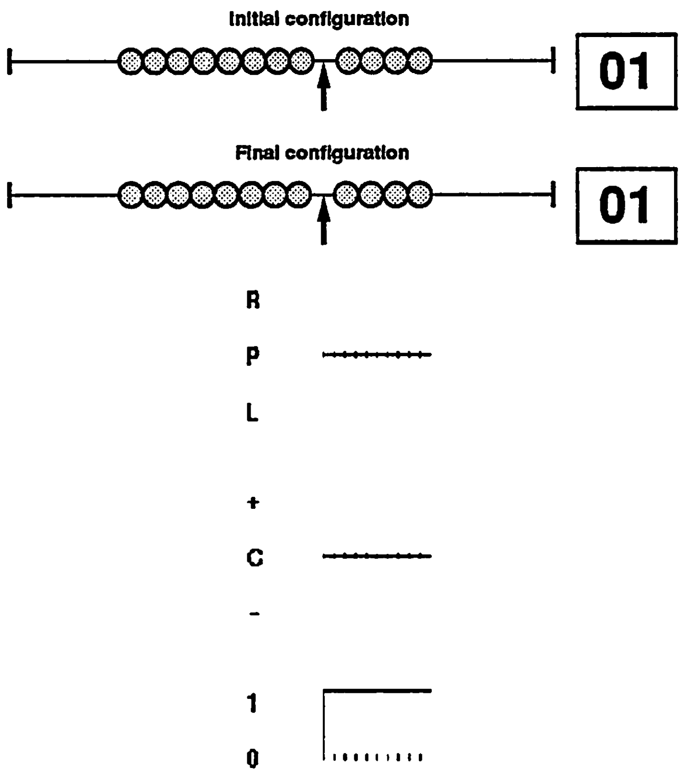


Figure 6.14. *Performance of the level 1 controller on the NOOP task.*

the multilevel controller was also unable to learn the count task even after 1,000,000 training runs. Compared to this, 5,000 training runs were sufficient to train each level of the multilevel controller when the command filter network was used.

Samples of the level 2 controller's performance in the COUNT task are shown in Figure 6.15. Instantiations of the abacus used in these tests were also generated randomly by picking 1 to 20 beads and by setting the counter to a random number between 0 and 10. In addition to showing the initial and final abacus configurations, the figure also shows the primitive actions output at each time step during the execution of the counting task. As these figures indicate, the level 2 controller learned to use one of many appropriate sequences of level 1 tasks to correctly perform the COUNT task. The sequence learned was LE–ZERO–RI, i.e., move the pointer to the left end, bring the counter down to zero, and then move one bead at a time to the right, incrementing the counter once for every bead. Moreover, depending on the initial configurations, the controller learned to execute only the necessary portions of this sequence.

6.4 Summary

In this chapter, we used two control problems as examples to demonstrate how learning to solve complex problems can be facilitated by the shaping process of first learning to solve simpler subproblems. Two different ways of implementing shaping were illustrated. In the first example, the behavior of a controller was shaped over time, while in the second, shaping was implemented through incremental development of the structure of the controller. In both examples, the shaping procedure proved indispensable for training the controller on the specified task.

Moreover, in the bead counting task, the use of a command filter module was also necessary for successfully training the controller. This provides experimental support for the heuristic upon which the idea of the command filter module was based.

Shaping is also a natural way of introducing domain knowledge in reinforcement learning systems. In the key pressing task, for example, domain knowledge helped determine the series of approximations used in the shaping procedure and the sensations to provide to the controller. Domain knowledge also helped determine what sensations to provide to each level of the controller in the bead counting task. Perhaps more tellingly, domain knowledge was used in the bead counting task to guide the crucial process of determining the sets of subtasks in the decomposition set. This obviates the difficulties involved in having the learning system determine the decomposition set.

Both these examples also attest to the viability of reinforcement learning approaches to learning complex control tasks. It has been the complaint of several researchers that direct reinforcement learning methods have been applied mostly to simple problems. We hope that the results presented here and in the previous chapters will serve as evidence of the broader applicability of these methods.

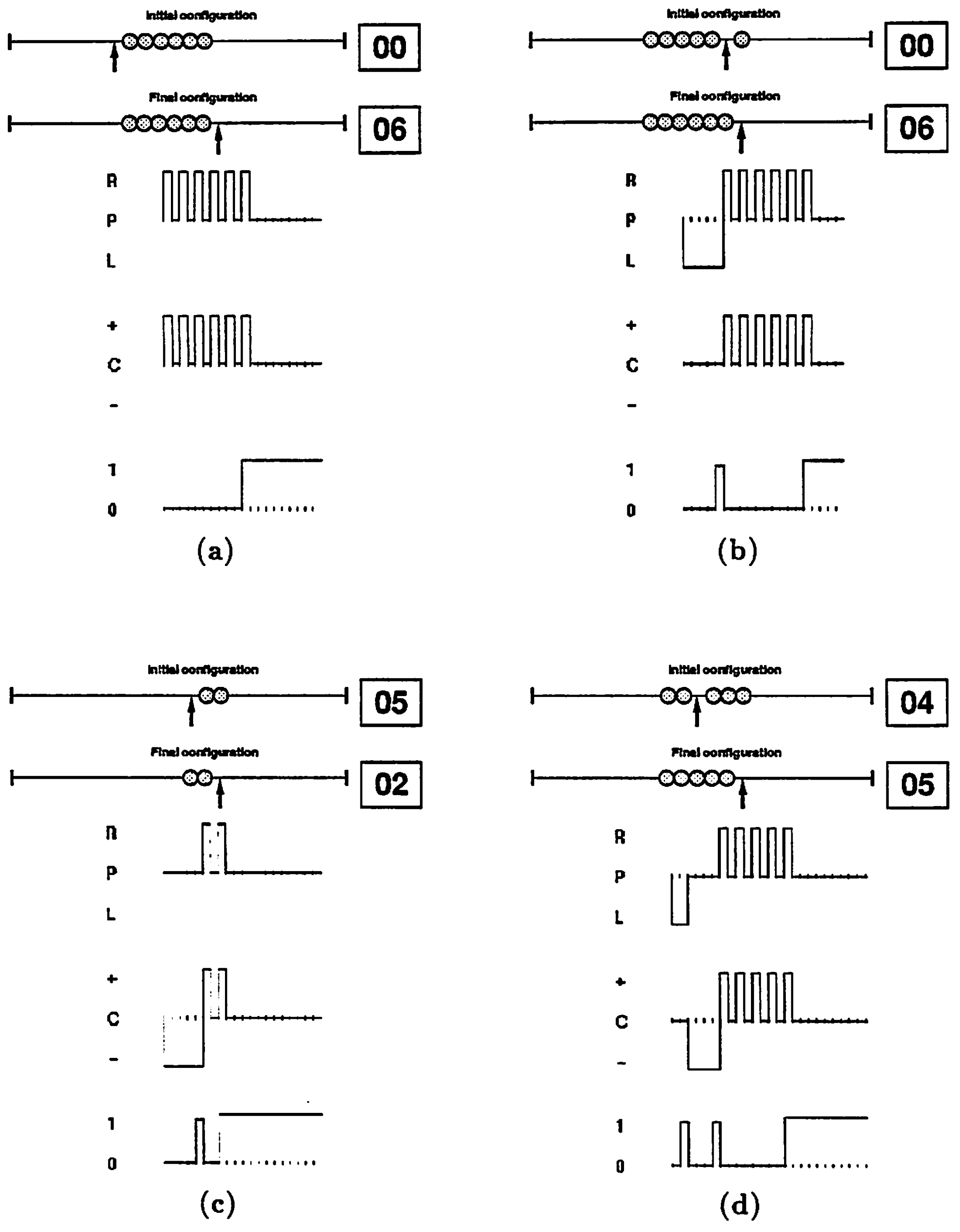


Figure 6.15. *Performance of the level 2 controller on the COUNT task for different initial abacus configurations.*

Chapter 7
Conclusions

In the introduction to this dissertation we asked the question: "...how useful are direct methods for training controllers...?" As a step toward answering this question, we demonstrated through several experiments that direct reinforcement learning methods can be used to train controllers to solve fairly difficult control problems. Included in these examples was a peg-in-hole insertion problem, which involved learning compliant control under real-world conditions of uncertainty and noise. Our results for this task indicate that direct reinforcement learning can be used to learn a reactive control strategy that works well even in the presence of a high degree of noise and uncertainty. Other experiments illustrated the capabilities of direct reinforcement learning methods in several different control problems.

Furthermore, by comparing the performance of the direct learning methods with that of indirect learning methods applied to the same control problems, we have argued in Chapter 4 that for some control problems, the requirement of modeling the index of performance can make indirect methods relatively inefficient. However, we have only scratched the surface in examining this issue. Much more work, both empirical and theoretical, is necessary to fully characterize the problems for which one type or the other of these learning methods—direct or indirect—is more effective.

We next considered problems in scaling direct methods to more complex reinforcement learning tasks. The first problem considered was that of structural credit assignment. As a solution to this problem, we presented a novel modular architecture that can be used for structural credit assignment in a reinforcement learning system. This architecture can be seen as a marriage of the best of direct and indirect methods and represents a compromise between the full-fledged models required by indirect methods and the complete absence of models in direct methods. Moreover, this approach also takes full advantage of the additional information available in control problems in the form of process output.

In Chapter 6, we addressed the question of whether having only evaluations available as training information—which is the case in reinforcement learning tasks—is a serious impediment to learning control. We demonstrated that by shaping the controller's behavior, i.e., by having the critic selects evaluation criteria judiciously based on the performance of the controller, and by designing the controller to take advantage of task decomposition where possible, we can train reinforcement learning controllers to perform fairly complex tasks. Shaping is a natural way to introduce domain knowledge into the training of a reinforcement learning system. Shaping has been used very effectively in training animals to perform complex tasks, and we feel that it has tremendous potential for training artificial learning systems.

Finally, we make the following observation about the implementation of reinforcement learning controllers. All the controllers in this dissertation were implemented using connectionist networks. Doing so underscored the necessity of reinforcement learning units that produce real-valued outputs because most of the control tasks required the use of real-valued control signals rather than binary signals. Our experiments demonstrate the utility of stochastic real-valued (SRV) units described in Chapter 3 and elsewhere [48] in such situations. We have also presented some new theoretical results obtained for the SRV algorithm, including a convergence theorem that applies to a simplified version of the SRV algorithm. Furthermore, the central idea of the SRV algorithm, namely, using predictions of the outcomes of *actions to* incrementally optimize the actions, is an important one that merits further study (see also [24, 136]).

In summary, in this dissertation, we have attempted to demonstrate the viability of direct reinforcement learning methods for learning control through several empirical studies. We have also presented and tested several approaches to scaling reinforcement learning to complex control problems. Our results indicate that contrary to popular belief, direct reinforcement learning methods can be extremely useful and efficient tools for learning control.

APPENDIX A

CONVERGENCE PROOF FOR THEOREM 1

Before proving Theorem 1 stated in Chapter 3, we make the following observation about the algorithm and establish three supporting lemmas.[1]

Observation Let A denote the matrix whose columns are the input vectors $\mathbf{x}^{(i)}, 1 \le i \le k$, and let $\mathrm{Ra}(A)$ denote the range of A. Then $\boldsymbol{\theta}_n \in \mathrm{Ra}(A) + \boldsymbol{\theta}_1, \;\; n \ge 1$. This is readily apparent if we write

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_1 + \sum_{j=1}^{n-1} \Delta\boldsymbol{\theta}_j, \;\; \text{where} \;\; \Delta\boldsymbol{\theta}_j = \boldsymbol{\theta}_{j+1} - \boldsymbol{\theta}_j,$$

and observe from (3.17) that each $\Delta\boldsymbol{\theta}_j$ is a scalar times $\mathbf{x}_j \in X$.

Lemma 1 Given an arbitrary finite $\boldsymbol{\theta}_1$ and $\boldsymbol{\beta} = (\beta^{(1)}, \beta^{(2)}, \ldots, \beta^{(k)})$ as in A3, there exists a unique $\boldsymbol{\theta}^* \in \mathrm{Ra}(A) + \boldsymbol{\theta}_1$ such that

$$\mu_i = \boldsymbol{\theta}^{*\mathsf{T}} \mathbf{x}^{(i)} = \beta^{(i)}, \; 1 \le i \le k. \tag{A.1}$$

Proof: Consider the equation

$$A^{\mathsf{T}} \boldsymbol{\theta} = \boldsymbol{\beta}, \tag{A.2}$$

where A is the matrix whose columns are the input vectors (as in the observation above). Since the rows of A are independent (from A2) there exists at least one $\boldsymbol{\theta}^*$ that satisfies (A.2). Also, since $\boldsymbol{\theta}^* \in \mathrm{Ra}(A) + \boldsymbol{\theta}_1$, we can rewrite (A.2) as $A^{\mathsf{T}}(A\boldsymbol{\omega} + \boldsymbol{\theta}_1) = \boldsymbol{\beta}$

[1]Supporting results similar to the above observation and Lemma 1 were used by Barto and Anandan [18] in their proof of convergence of the A_{R-P} algorithm.

for some $\omega \in \Re^k$. Since the columns of A are independent, $A^\top A$ is invertible, and ω is the unique vector

$$\omega = (A^\top A)^{-1}(\beta - A^\top \theta_1). \tag{A.3}$$

Thus $\theta^* = A\omega + \theta_1$, with ω defined by (A.3), is the unique vector in $\mathrm{Ra}(A) + \theta_1$ that satisfies (A.1). □

Lemma 2

$$E\{g_n(\theta_n, \mathbf{x}_n) \mid \theta_n, \mathbf{x}_n\} = \sigma_n R(\theta_n^\top \mathbf{x}_n, \mathbf{x}_n). \tag{A.4}$$

Proof: From (3.18),

$$g_n(\theta_n, \mathbf{x}_n) = (r(y_n, \mathbf{x}_n) - r(\mu_n, \mathbf{x}_n)) \left(\frac{y_n - \mu_n}{\sigma_n}\right),$$

where $y_n \sim N(\mu_n, \sigma_n)$ is a Gaussian random variable and $\mu_n = \theta_n^\top \mathbf{x}_n$. Taking conditional expectations on both sides, we get

$$\begin{aligned} &E\{g_n(\theta_n, \mathbf{x}_n) \mid \theta_n, \mathbf{x}_n\} \\ &\quad = E\left\{r(y_n, \mathbf{x}_n)\left(\frac{y_n - \mu_n}{\sigma_n}\right) \mid \theta_n, \mathbf{x}_n\right\} - E\left\{r(\mu_n, \mathbf{x}_n)\left(\frac{y_n - \mu_n}{\sigma_n}\right) \mid \theta_n, \mathbf{x}_n\right\} \\ &\quad = E\left\{r(y_n, \mathbf{x}_n)\left(\frac{y_n - \mu_n}{\sigma_n}\right) \mid \theta_n, \mathbf{x}_n\right\} \qquad (A.5) \end{aligned}$$

because $r(\mu_n, \mathbf{x}_n)$ and $\left(\frac{y_n - \mu_n}{\sigma_n}\right)$ are conditionally independent, given θ_n and $\mathbf{x}_n$, and $E\left\{\left(\frac{y_n - \mu_n}{\sigma_n}\right) \mid \theta_n, \mathbf{x}_n\right\} = 0$. But

$$\begin{aligned} &E\left\{r(y_n, \mathbf{x}_n)\left(\frac{y_n - \mu_n}{\sigma_n}\right) \mid \theta_n, \mathbf{x}_n\right\} \\ &\quad = \int_{-\infty}^{\infty} E\left\{r(y_n, \mathbf{x}_n)\left(\frac{y_n - \mu_n}{\sigma_n}\right) \mid \theta_n, \mathbf{x}_n, y_n\right\} dD_n(y_n \mid \theta_n, \mathbf{x}_n), \end{aligned}$$

where $D_n(.)$ is the distribution function for y_n. Therefore

$$\begin{aligned} E\left\{r(y_n, \mathbf{x}_n)\left(\frac{y_n - \mu_n}{\sigma_n}\right) \mid \theta_n, \mathbf{x}_n\right\} &= \int_{-\infty}^{\infty} M(y_n, \mathbf{x}_n)\left(\frac{y_n - \mu_n}{\sigma_n}\right) dD_n(y_n \mid \theta_n, \mathbf{x}_n) \\ &= E\left\{M(y_n, \mathbf{x}_n)\left(\frac{y_n - \mu_n}{\sigma_n}\right) \mid \theta_n, \mathbf{x}_n\right\}. \qquad (A.6) \end{aligned}$$

Substituting the result (A.6) in (A.5), we get

$$\begin{aligned}
&E\left\{g_n(\theta_n, \mathrm{x}_n) \mid \theta_n, \mathrm{x}_n\right\} \\
&\quad = E\left\{M(y_n, \mathrm{x}_n)\left(\frac{y_n-\mu_n}{\sigma_n}\right) \mid \theta_n, \mathrm{x}_n\right\} \\
&\quad = E\left\{\left[M(\mu_n, \mathrm{x}_n) + \frac{\partial M(\mu_n, \mathrm{x}_n)}{\partial y}(y_n-\mu_n) + \frac{\partial^2 M(\zeta_n, \mathrm{x}_n)}{\partial y^2}\frac{(y_n-\mu_n)^2}{2}\right]\right. \\
&\qquad\qquad \left.\left(\frac{y_n-\mu_n}{\sigma_n}\right) \mid \theta_n, \mathrm{x}_n\right\},
\end{aligned}$$

using a second order Taylor series expansion of $M(y_n, \mathrm{x}_n)$ about μ_n (ζ_n lies between y_n and μ_n). Therefore

$$\begin{aligned}
&E\left\{g_n(\theta_n, \mathrm{x}_n) \mid \theta_n, \mathrm{x}_n\right\} \\
&\quad = M(\mu_n, \mathrm{x}_n) E\left\{\left(\frac{y_n-\mu_n}{\sigma_n}\right) \mid \theta_n, \mathrm{x}_n\right\} + R(\mu_n, \mathrm{x}_n) E\left\{\left(\frac{(y_n-\mu_n)^2}{\sigma_n}\right) \mid \theta_n, \mathrm{x}_n\right\} \\
&\qquad + \frac{1}{2} E\left\{\frac{\partial^2 M(\zeta_n, \mathrm{x}_n)}{\partial y^2}\left(\frac{(y_n-\mu_n)^3}{\sigma_n}\right) \mid \theta_n, \mathrm{x}_n\right\}.
\end{aligned}$$

The first term on the rhs above is zero because the odd moments of y_n, a Gaussian random variable, are zero. Since $M(.)$ has bounded second order derivatives by assumption A3, the last term on the rhs can also be seen to be zero. Hence we get

$$E\left\{g_n(\theta_n, \mathrm{x}_n) \mid \theta_n, \mathrm{x}_n\right\} = \sigma_n R(\mu_n, \mathrm{x}_n) = \sigma_n R(\theta_n^{\mathsf{T}} \mathrm{x}_n, \mathrm{x}_n).$$

□

Lemma 3

$$E\left\{\|g_n(\theta_n, \mathrm{x}_n)\|^2 \mid \theta_n, \mathrm{x}_n\right\} \leq K(1 + \|\theta_n - \theta^*\|^2). \qquad \text{(A.7)}$$

Proof: From (3.18)

$$\begin{aligned}
&E\left\{\|g_n(\theta_n, \mathrm{x}_n)\|^2 \mid \theta_n, \mathrm{x}_n\right\} \\
&\quad = E\left\{(r(y_n, \mathrm{x}_n) - r(\mu_n, \mathrm{x}_n))^2 \left(\frac{y_n-\mu_n}{\sigma_n}\right)^2 \mid \theta_n, \mathrm{x}_n\right\}
\end{aligned}$$

$$= \int_{-\infty}^{\infty} E\left\{(r(y_n, \mathrm{x}_n) - r(\mu_n, \mathrm{x}_n))^2 \mid \theta_n, \mathrm{x}_n, y_n\right\}\left(\frac{y_n - \mu_n}{\sigma_n}\right)^2 dD_n(y_n \mid \theta_n, \mathrm{x}_n). \quad (A.8)$$

Now

$$\begin{aligned} &E\left\{(r(y_n, \mathrm{x}_n) - r(\mu_n, \mathrm{x}_n))^2 \mid \theta_n, \mathrm{x}_n, y_n\right\} \\ &\qquad = E\left\{r^2(y_n, \mathrm{x}_n) \mid \theta_n, \mathrm{x}_n, y_n\right\} + E\left\{r^2(\mu_n, \mathrm{x}_n) \mid \theta_n, \mathrm{x}_n, y_n\right\} \\ &\qquad\qquad - 2E\left\{r(y_n, \mathrm{x}_n) r(\mu_n, \mathrm{x}_n) \mid \theta_n, \mathrm{x}_n, y_n\right\}. \end{aligned}$$

Defining $\beta_n = \beta^{(i)}$ if $\mathrm{x}_n = \mathrm{x}^{(i)} \in X$ (i.e., $\beta_n = \theta^{*\mathsf{T}}\mathrm{x}_n$), we can use assumption A5 and the conditional independence a.e. of $r(y_n, \mathrm{x}_n)$ and $r(\mu_n, \mathrm{x}_n)$, given θ_n, x_n, and y_n, in the above equation to obtain

$$\begin{aligned} &E\left\{(r(y_n, \mathrm{x}_n) - r(\mu_n, \mathrm{x}_n))^2 \mid \theta_n, \mathrm{x}_n, y_n\right\} \\ &\qquad \leq [h(1 + (y_n - \beta_n)^2 + M^2(y_n, \mathrm{x}_n)] + [h(1 + (\mu_n - \beta_n)^2 + M^2(\mu_n, \mathrm{x}_n)] \\ &\qquad\qquad - 2M(y_n, \mathrm{x}_n) M(\mu_n, \mathrm{x}_n) \\ &\qquad = h(2 + (y_n - \beta_n)^2 + (\mu_n - \beta_n)^2) + (M(y_n, \mathrm{x}_n) - M(\mu_n, \mathrm{x}_n))^2. \quad (A.9) \end{aligned}$$

Substituting (A.9) in (A.8), we get

$$\begin{aligned} &E\left\{\|g_n(\theta_n, \mathrm{x}_n)\|^2 \mid \theta_n, \mathrm{x}_n\right\} \\ &\qquad \leq 2h \int_{-\infty}^{\infty} \left(\frac{y_n - \mu_n}{\sigma_n}\right)^2 dD_n(y_n \mid \theta_n, \mathrm{x}_n) \\ &\qquad\qquad + h \int_{-\infty}^{\infty} \left((y_n - \beta_n)^2 + (\mu_n - \beta_n)^2\right)\left(\frac{y_n - \mu_n}{\sigma_n}\right)^2 dD_n(y_n \mid \theta_n, \mathrm{x}_n) \\ &\qquad\qquad + \int_{-\infty}^{\infty} (M(y_n, \mathrm{x}_n) - M(\mu_n, \mathrm{x}_n))^2 \left(\frac{y_n - \mu_n}{\sigma_n}\right)^2 dD_n(y_n \mid \theta_n, \mathrm{x}_n). \end{aligned}$$

Using the fact that $y_n \sim N(\mu_n, \sigma_n)$ and expanding $M(y_n, \mathrm{x}_n)$ in a second order Taylor series about μ_n as before, we get

$$\begin{aligned} &E\left\{\|g_n(\theta_n, \mathrm{x}_n)\|^2 \mid \theta_n, \mathrm{x}_n\right\} \\ &\quad \leq 2h + h \int_{-\infty}^{\infty} [(y_n - \mu_n)^2 + 2(y_n - \mu_n)(\mu_n - \beta_n) + 2(\mu_n - \beta_n)^2] \left(\frac{y_n - \mu_n}{\sigma_n}\right)^2 dD_n(.) \end{aligned}$$

$$+\int_{-\infty}^{\infty}\left[\frac{\partial M(\mu_n,\mathrm{x}_n)}{\partial y}(y_n-\mu_n)+\frac{\partial^2 M(\zeta_n,\mathrm{x}_n)}{\partial y^2}\frac{(y_n-\mu_n)^2}{2}\right]^2\left(\frac{y_n-\mu_n}{\sigma_n}\right)^2 dD_n(.)$$

$$= \quad 2h+h\int_{-\infty}^{\infty}\frac{(y_n-\mu_n)^4}{\sigma_n^2}dD_n(.)+2h(\mu_n-\beta_n)\int_{-\infty}^{\infty}\frac{(y_n-\mu_n)^3}{\sigma_n^2}dD_n(.)$$

$$+2h(\mu_n-\beta_n)^2\int_{-\infty}^{\infty}\frac{(y_n-\mu_n)^2}{\sigma_n^2}dD_n(.)+R^2(.)\int_{-\infty}^{\infty}\frac{(y_n-\mu_n)^4}{\sigma_n^2}dD_n(.)$$

$$+\int_{-\infty}^{\infty}\left(\frac{\partial^2 M(\zeta_n,\mathrm{x}_n)}{\partial y^2}\right)^2\frac{(y_n-\mu_n)^6}{4\sigma_n^2}dD_n(.)$$

$$+R(.)\int_{-\infty}^{\infty}\left(\frac{\partial^2 M(\zeta_n,\mathrm{x}_n)}{\partial y^2}\right)\frac{(y_n-\mu_n)^5}{\sigma_n^2}dD_n(.).$$

Since the second order derivatives of $M(.)$ are bounded by assumption A3, these derivatives can be replaced in the last two terms of the rhs above by an upper bound S and factored out of the integrals without affecting the inequality. The integrals on the rhs are then all moments of y_n, which can be easily evaluated because y_n has a Gaussian distribution. Therefore

$$\begin{aligned}
&E\left\{\|g_n(\theta_n,\mathrm{x}_n)\|^2 \mid \theta_n,\mathrm{x}_n\right\}\\
&\quad\leq\; 2h+3h\sigma_n^2+2h(\mu_n-\beta_n)^2+3R^2(.)\sigma_n^2+\frac{15}{4}S^2\sigma_n^4\\
&\quad\leq\; H(1+(\mu_n-\beta_n)^2)
\end{aligned}$$

for some $H>0$, by the boundedness of $\sigma_n, R(\mu_n,\mathrm{x}_n)$, and $\frac{\partial^2 M(.)}{\partial y^2}$. But

$$(\mu_n-\beta_n)^2=(\theta_n^{\mathsf{T}}\mathrm{x}_n-\theta^{*\mathsf{T}}\mathrm{x}_n)^2=((\theta_n-\theta^*)^{\mathsf{T}}\mathrm{x}_n)^2\leq\|\theta_n-\theta^*\|^2\,\|\mathrm{x}_n\|^2$$

by Cauchy's inequality. Substituting for $(\mu_n-\beta_n)^2$ in the above, we get (because $\mathrm{x}_n\in X$, a finite set), for some $K>0$,

$$E\left\{\|g_n(\theta_n,\mathrm{x}_n)\|^2 \mid \theta_n,\mathrm{x}_n\right\}\leq K(1+\|\theta_n-\theta^*\|^2).$$

□

We are now ready to prove Theorem 1 stated in Chapter 3. Our proof of this theorem is based on Gladyshev's proof of the convergence of the Robbins-Monro process [44]. Unlike the Robbins-Monro process, wherein the stochasticity is confined to the environment in which the learning system operates, (3.14) defines a *stochastic* system operating in a stochastic environment. Confounding between these two sources of randomness makes the analysis more complicated. Hence, although we employed the same basic proof technique based on Martingale theory as Gladyshev, his proof had to be extended in several non-trivial ways. Other alternative approaches could have been taken to arrive at the same result but we felt that the approach presented here is most easily comprehensible.

Proof of Theorem 1: Let

$$\mathbf{e}_n = (\boldsymbol{\theta}_n - \boldsymbol{\theta}^*). \tag{A.10}$$

Clearly, from (3.17),

$$\mathbf{e}_{n+1} = \mathbf{e}_n + \sigma_n^2 g_n(\boldsymbol{\theta}_n, \mathbf{x}_n)\,\mathbf{x}_n. \tag{A.11}$$

Squaring and taking conditional expectations given $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n$, we get

$$\begin{aligned} E\left\{\|\mathbf{e}_{n+1})\|^2 \mid \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n\right\} &= E\left\{\|\mathbf{e}_n)\|^2 \mid \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n\right\} \\ &\quad + \sigma_n^4 E\left\{\|g_n(\boldsymbol{\theta}_n, \mathbf{x}_n)\mathbf{x}_n\|^2 \mid \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n\right\} \\ &\quad + 2\sigma_n^2 E\left\{\mathbf{e}_n^{\mathsf{T}} g_n(\boldsymbol{\theta}_n, \mathbf{x}_n)\mathbf{x}_n \mid \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n\right\}. \end{aligned} \tag{A.12}$$

In this equation,

$$\begin{aligned} &E\left\{\|g_n(\boldsymbol{\theta}_n, \mathbf{x}_n)\mathbf{x}_n\|^2 \mid \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n\right\} \\ &\qquad = \sum_{i=1}^{k} \xi_i E\left\{\|g_n(\boldsymbol{\theta}_n, \mathbf{x}^{(i)})\mathbf{x}^{(i)}\|^2 \mid \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n, \mathbf{x}^{(i)}\right\} \\ &\qquad = \sum_{i=1}^{k} \xi_i E\left\{\|g_n(\boldsymbol{\theta}_n, \mathbf{x}^{(i)})\|^2 \mid \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n, \mathbf{x}^{(i)}\right\} \|\mathbf{x}^{(i)}\|^2 \\ &\qquad \le \sum_{i=1}^{k} K(1 + \|\mathbf{e}_n\|^2)\xi^{(i)}\|\mathbf{x}^{(i)}\|^2 \quad \text{(using Lemma 3)} \end{aligned}$$

$$\leq \quad K_1(1 + \|e_n\|^2), \tag{A.13}$$

where $K_1 = kK \max_{1\leq i\leq k} \xi^{(i)}\|x^{(i)}\|^2$. Also

$$\begin{aligned}
&E\left\{e_n^{\top} g_n(\theta_n, x_n)x_n \mid \theta_1, \ldots, \theta_n\right\} \\
&\quad = \sum_{i=1}^{k} E\left\{e_n^{\top} g_n(\theta_n, x^{(i)})x^{(i)} \mid \theta_n, x^{(i)}\right\} \xi^{(i)} \\
&\quad = \sum_{i=1}^{k} E\left\{g_n(\theta_n, x^{(i)}) \mid \theta_n, x^{(i)}\right\} e_n^{\top} x^{(i)}\xi^{(i)} \qquad (A.14) \\
&\quad = \sigma_n \sum_{i=1}^{k} R(\theta_n^{\top} x^{(i)}, x^{(i)})(\theta_n - \theta^*)^{\top} x^{(i)}\xi^{(i)} \\
&\quad = \sigma_n \sum_{i=1}^{k} R(\theta_n^{\top} x^{(i)}, x^{(i)})(\theta_n^{\top} x^{(i)} - \beta^{(i)})\xi^{(i)} \\
&\quad \leq 0 \text{ (by assumption A4).} \qquad (A.15)
\end{aligned}$$

Using (A.13) and (A.15) in (A.12), we get

$$\begin{aligned}
E\left\{\|e_{n+1})\|^2 \mid \theta_1, \ldots, \theta_n\right\} &\leq \|e_n\|^2 + \sigma_n^4 K_1(1 + \|e_n\|^2) \\
&= \|e_n\|^2(1 + K_1\sigma_n^4) + K_1\sigma_n^4. \qquad (A.16)
\end{aligned}$$

Let us define

$$\alpha_n = \|e_n\|^2 \prod_{j=n}^{\infty}(1 + K_1\sigma_j^4) + \sum_{j=n}^{\infty} K_1\sigma_j^4 \prod_{i=j+1}^{\infty} (1 + K_1\sigma_i^4). \tag{A.17}$$

Then it is easy to show using (A.16) that

$$E\left\{\alpha_{n+1} \mid \theta_1, \ldots, \theta_n\right\} \leq \alpha_n. \tag{A.18}$$

Taking conditional expectations for given $\alpha_1, \ldots, \alpha_n$ on both sides of the inequality (A.18), we find

$$E\left\{\alpha_{n+1} \mid \alpha_1, \ldots, \alpha_n\right\} \leq \alpha_n,$$

which shows that α_n is a non-negative supermartingale where

$$E\{\alpha_{n+1}\} \leq E\{\alpha_n\} \leq \cdots \leq E\{\alpha_1\} < \infty. \tag{A.19}$$

Therefore, by the martingale convergence theorem [31], α_n converges with probability 1. From this observation, the definition of α_n (Equation (A.17)) and assumption A1, we can conclude that $||\mathbf{e}_n||^2 \xrightarrow{\text{w.p.1}} \eta$, a random variable. Also, (A.19) and (A.17) together with A1 imply that

$$E\{||\mathbf{e}_n||^2\} < \infty. \tag{A.20}$$

Let us now take expectations on both sides of (A.12) after making substitutions using (A.13) and (A.14). We get

$$E\{||\mathbf{e}_{n+1}||^2\} - E\{||\mathbf{e}_n||^2\} \leq \sigma_n^4 K_1(1+E\{||\mathbf{e}_n||^2\}) + 2\sigma_n^3 E\left\{\sum_{i=1}^{k} R(\boldsymbol{\theta}_n^{\mathsf{T}}\mathbf{x}^{(i)}, \mathbf{x}^{(i)})\mathbf{e}_n^{\mathsf{T}}\mathbf{x}^{(i)}\xi^{(i)}\right\}. \tag{A.21}$$

Adding the first n of these inequalities, we get

$$\begin{aligned} E\{||\mathbf{e}_{n+1}||^2\} - E\{||\mathbf{e}_1||^2\} \;\; \leq \;\; & \sum_{j=1}^{n} \sigma_j^4 K_1(1+E\{||\mathbf{e}_j||^2\}) \\ & + 2\sum_{j=1}^{n} \sigma_j^3 E\left\{\sum_{i=1}^{k} R(\boldsymbol{\theta}_j^{\mathsf{T}}\mathbf{x}^{(i)}, \mathbf{x}^{(i)})\mathbf{e}_j^{\mathsf{T}}\mathbf{x}^{(i)}\xi^{(i)}\right\}. \end{aligned} \tag{A.22}$$

From inequality (A.22), using the boundedness of $E\{||\mathbf{e}_n||^2\}$ and assumption A1, it follows that

$$\sum_{j=1}^{\infty} \sigma_j^3 E\left\{\sum_{i=1}^{k} R(\boldsymbol{\theta}_j^{\mathsf{T}}\mathbf{x}^{(i)}, \mathbf{x}^{(i)})\mathbf{e}_j^{\mathsf{T}}\mathbf{x}^{(i)}\xi^{(i)}\right\} > -\infty, \tag{A.23}$$

which implies that

$$\sum_{j=1}^{\infty} \sigma_j^3 E\left\{\sum_{i=1}^{k} -R(\boldsymbol{\theta}_j^{\mathsf{T}}\mathbf{x}^{(i)}, \mathbf{x}^{(i)})\mathbf{e}_j^{\mathsf{T}}\mathbf{x}^{(i)}\xi^{(i)}\right\} < \infty. \tag{A.24}$$

Since $\sum_{j=1}^{\infty} \sigma_j^3$ diverges (A1) and, by A4 and A2, the quantity

$$\sum_{i=1}^{k} -R(\boldsymbol{\theta}_j^{\mathsf{T}}\mathbf{x}^{(i)}, \mathbf{x}^{(i)})\mathbf{e}_j^{\mathsf{T}}\mathbf{x}^{(i)}\xi^{(i)}$$

is non-negative, we can conclude that for some subsequence $\{n_j\}$,

$$\sum_{i=1}^{k} -R(\boldsymbol{\theta}_{n_j}^{\mathsf{T}}\mathbf{x}^{(i)}, \mathbf{x}^{(i)})\mathbf{e}_{n_j}^{\mathsf{T}}\mathbf{x}^{(i)}\xi^{(i)} \to 0 \text{ w.p.1.}$$

Since each of the terms of the above sum is non-negative (A4), we have

$$R(\boldsymbol{\theta}_{n_j}^{\mathsf{T}}\mathbf{x}^{(i)}, \mathbf{x}^{(i)})\mathbf{e}_{n_j}^{\mathsf{T}}\mathbf{x}^{(i)} \to 0 \text{ w.p.1 } \forall\, 1 \leq i \leq k. \tag{A.25}$$

The fact that $\|\mathbf{e}_n\|^2 \to \eta$ w.p.1, together with assumptions A2, A3, and A4 and (A.25) imply that $\eta = 0$ w.p.1. Hence, $\boldsymbol{\theta}_n \to \boldsymbol{\theta}^*$ w.p.1. □

Appendix B

Simulation Details for Chapters 4, 5, and 6

This appendix provides additional details on the simulations reported in Chapters 4 through 6. For each simulation, we describe the controlled process, the connectionist network used to implement the learning controller, and the evaluation function used. The networks used in these simulations were all standard fully-connected feedforward connectionist networks (see [116, 58] or Section 1.3). Several different types of units were used in these networks. These are described briefly in the next section. The connectionist terminology used to describe these units and networks is defined in Section 1.3.

B.1 Types of units used in controller networks

A_{R-P} units Associative reward-penalty units were developed by Barto and Anandan [18]. These units are useful for learning binary-valued outputs via reinforcement feedback.

Backpropagation units These units use a differentiable nonlinear function (for example, the logistic function in Equation (3.25)) to compute their outputs. As a result, networks of these units can be used to learn nonlinear input–output mappings. The backpropagation algorithm used to adapt their weights was developed independently by several researchers [80, 111, 116, 143]. The most popular description of the backpropagation algorithm is that of Rumelhart, Hinton, and Williams [116].

Binary units Also called Perceptrons [114], or Threshold Logic Units (TLUs), these units produce binary outputs by thresholding the weighted sum of inputs. Various algorithms can be used to adapt their weights depending on the function that is to be learned.

LMS units Least Mean Square (LMS) units are linear units trained using the Widrow-Hoff rule [148].

RLS units Recursive Least Squares (RLS), units are also linear units but their weights are adapted using a different algorithm called the Recursive Least Squares algorithm (e.g., [84]). Although RLS units require more memory for implementation when compared with LMS units, RLS units can exhibit faster learning than the LMS units.

SRV units Stochastic real-valued units are described in detail in Chapter 3 and in [48]. These units are designed to learn real-valued outputs through reinforcement feedback.

The initial weights of all LMS and backpropagation units used in our networks were set to random values selected from a uniform distribution over the interval $[-0.5, 0.5]$. The initial weights of the rest of the units were set to 0. All the units except the RLS unit have a learning rate parameter, α. In addition, the A_{R-P} units have a penalty discount factor, λ, (see [18]), and the SRV units have a additional learning rate parameter, ρ, for adapting the parameter vector ϕ (see Equation 3.13).

B.2 Simulations in Chapter 4

Peg-in-hole insertion

The process We used the Zebra Zero robot produced by Zebra Robotics Inc., Palo Alto, CA, to perform peg-in-hole insertions. The peg used for this task was a wooden dowel, $22.225mm$ ($7/8in$) in diameter and $50mm$ long, affixed to the gripper of the robot. Since we were addressing a two-dimensional version of the peg-in-hole insertion task, a 23.8125*mm* (15/16*in*) wide slot—obtained by fastening two "L" shaped metal plates together with spacers in between—was used as the hole.

The controller The network used for the controller is described in Section 4.1. α was set to 0.1 for the backpropagation units and 0.05 for the output (SRV) units. In addition, ρ was also set to 0.05 for the SRV units.

The evaluation The position of the peg (X, Y, Θ) and the force and moment sensations (F_x, F_y, M_z) were used to compute the evaluation, r, as

$$r = \begin{cases} 1.0 - 0.01(|X| + |Y|) - |\Theta| & \text{if all forces are } \leq 0.5N, \\ 1.0 - 0.01(|X| + |Y|) - |\Theta| - 0.1F_{\max} & \text{otherwise,} \end{cases}$$

where $F_{\max}$ denotes the largest magnitude force component. If the evaluation computed above was less than zero, it was set to zero.

Inverse kinematics

The process The planar arm used in the inverse kinematics task had three rotating links (see Figure 4.8). The equation governing the relationship between the joint angles, $(\theta_1, \theta_2, \theta_3)$, and the position of the end-effector, (x, y), is the following:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3) \\ l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) + l_3 \sin(\theta_1 + \theta_2 + \theta_3) \end{bmatrix},$$

where l_1, l_2, and l_3 are the lengths of the links. In our simulations, all three link lengths were set to 0.3333333 units. The first joint of the arm was located at $(0, 0)$, and the three target buttons were located at $(-0.25, 0.25)$, $(0.25, 0.25)$, and $(0.0, 0.65)$.

The controller The controller network used for this task had 2 input, 50 hidden, and 3 output units. When the direct method was used, the hidden units used were backpropagation units and the output units used were SRV units. α was set to 0.95 for the output units and to 0.1 for the hidden units. In addition, ρ was set to 0.76 for the output units.

With the indirect method, both the hidden and the output units of the controller were backpropagation units, and in addition, a forward model consisting of 3 input, 50 hidden, and 2 output units—all backpropagation units—was used. In this case, α was set to 0.1 for all the units.

The evaluation The evaluation, r, was computed using the error between the cartesian position of the end-effector, (x, y), and the center of the target button, (x_t, y_t) as

$$r = \max\left(0.0,\ 1.0 - \left(\frac{(x_t - x)^2 + (y_t - y)^2}{2}\right)^{1/2}\right).$$

The cart-pole problem

The process The following non-linear differential equations were used to model the cart-pole system depicted in Figure 4.10:

$$\begin{aligned}\ddot{x} &= \frac{F + ml\left[\dot{\theta}^2 \sin\theta - \ddot{\theta}\cos\theta\right] - \mu_c \operatorname{sgn}(\dot{x})}{M + m} \\ \ddot{\theta} &= \frac{g \sin\theta - \ddot{x}\cos\theta - \frac{\mu_p \dot{\theta}}{ml}}{l\left[\frac{4}{3} - \frac{m\cos^2\theta}{M+m}\right]}.\end{aligned}$$

Here $M = 1.0kg$ and $m = 0.1kg$ are the masses of the cart and the pole respectively, $l = 0.5m$ is *half* the length of the pole, $\mu_p = 0.000002$ is the coefficient of friction of pole on cart, $\mu_c = 0.0005$ is the coefficient of friction of cart on track, $g = -9.8m/s^2$ is the acceleration due to gravity, and F is the force applied to the cart. A fourth order Runge-Kutta method with a time step of $0.01s$ was used to approximate numerically the solution of these equations. Moreover, in these simulations, the track length was $4.8m$.

The controller The network used for the controller is described in detail in Section 4.3. The learning rate α was set to 0.05 for the controller and 0.3 for the internal critic network except on the connections from the controller, where it was 0.9. For the SRV unit used in the direct method, ρ was set to 0.3.

The evaluation The critic signaled failure whenever the cart hit the end of the track ($|\, x \,| > 2.4m$) or whenever the pole fell ($|\, \theta \,| > 12°$).

B.3 Simulations in Chapter 5

The interference task

The process The abstract "process" used for the interference task is defined by the entries in the third column of Table 5.1.

The controller The controller for this task was implemented as a single layer of n SRV units ($n = 2, 3, \text{or } 4$). Both α and ρ were set to 0.1 when n equaled 2 and 0.08 when n equaled 3 or 4.

In addition, when the LLM method was used for structural credit assignment, a single RLS unit was used to implement the LLM, and $n + 1$ linear units were used to implement the memory module. For a given input, the outputs of n of these linear units were used as the weights between the n SRV units and the RLS unit and the output of the one extra linear unit was used as the bias weight of the RLS unit. The weight updates for the RLS unit determined by the RLS algorithm were used as output errors for training the linear units of the memory module, for which the learning rate, α, was set to 1.0.

The evaluation The evaluation, r, returned by the critic in the interference task depended on the error, e, defined in the fourth column of Table 5.1. Specifically,

$$r = \begin{cases} 1.0 - 30e & \text{if } e < 0.03, \\ 0.1 - 0.1e & \text{otherwise,} \end{cases}$$

B.4 Simulations in Chapter 6

The key-pressing task

The process As described in Section 6.2.1, the controlled process in the key-pressing task was a dynamically simulated Stanford/JPL hand (see Figure 6.1). Only the index finger of the hand was used. Furthermore, the motion of the hand base was restricted to a plane parallel to the x-y plane. The control actions were positioning commands that moved the hand base in its plane and positioned the 3 joints of the index finger.

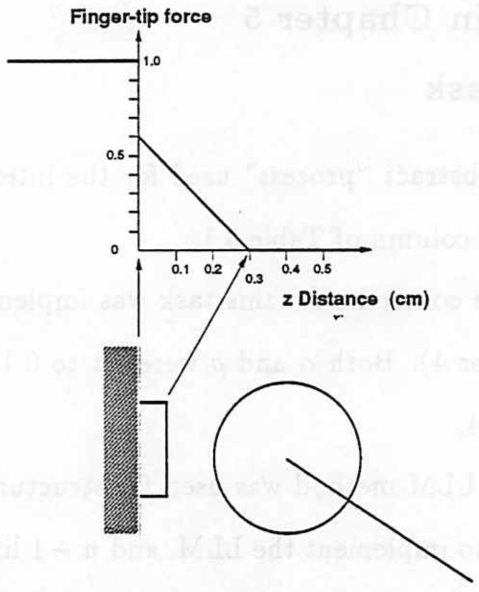


Figure B.1. *The output of the simulated fingertip force sensor as a function of the height of the fingertip above the keypad.*

Proprioceptive feedback of the position and velocity of each joint was provided to the controller. Additionally, a simulated fingertip force sensation was used to provide feedback during key presses. This force was a function of the height of the fingertip above the keypad, as shown in Figure B.1.

The controller The architecture of the controller is shown in Figure B.2. The inputs to the network included a 24-bit command input specifying the target key, feedback of the position and velocity of the hand and the force on the fingertip, and also an efference copy of the network's output at the previous time step. The network had 30 backpropagation units in the hidden layer and 5 SRV units in the output layer. As shown in the figure, the output units are connected to both the input and the hidden units. α was set to 0.01 for the hidden units and 0.001 for the SRV units. ρ was also set to 0.001 for the SRV units.

The evaluation In each training run, the evaluation was computed using a sequence of criteria based on the portion of the key-press operation already accomplished (see Section 6.2.1). For the purposes of computing the evaluation, the fingertip was considered to be positioned over the target key when the x-y distance to the key

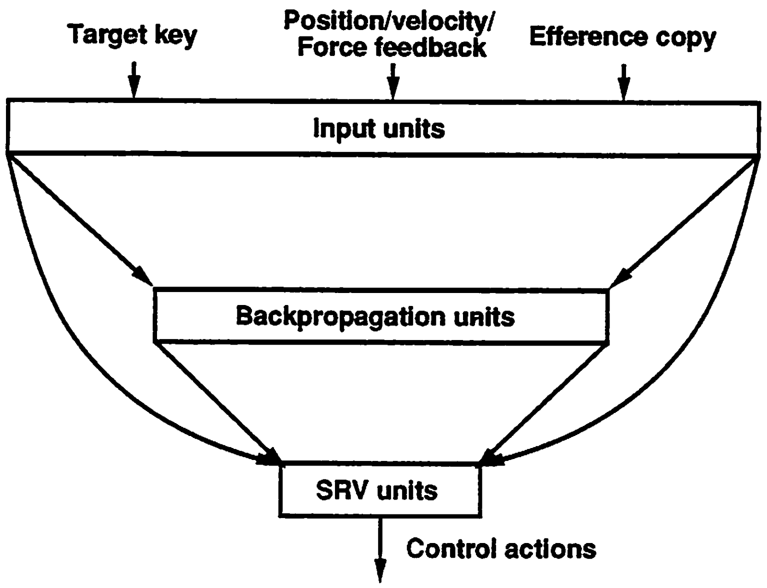


Figure B.2. *The architecture of the controller used in the key-pressing task.*

(denoted d) became zero and the hand velocities became very small. The key was considered to be pressed when the fingertip height (denoted z) became 0. In the following, the height of the fingertip at the previous time step is denoted z_{t-1}. Also note that each key on the keypad is $0.3cm$ high (see Figure B.1).

The procedure used for computing the evaluation is the following. If the fingertip is not yet positioned over the target key,

$$r = \begin{cases} 0.5\left(1 - \frac{|1-z|}{6}\right) & \text{if } z < 0.3, \\ 0.5\left(e^{-d/5} - \frac{|1-z|}{6}\right) & \text{otherwise.} \end{cases}$$

Once the fingertip is positioned over the target key,

$$r = \begin{cases} 0.7r_{\text{PRESS}} + 0.3d & \text{if key has NOT been pressed,} \\ r_{\text{RELEASE}} & \text{otherwise,} \end{cases}$$

where

$$r_{\text{PRESS}} = \begin{cases} \left\{ \begin{array}{ll} 0.5\left(1 - \frac{z}{3}\right) + 0.5 & \text{if } z < z_{t-1}, \\ 0.5\left(1 - \frac{z}{3}\right) & \text{otherwise} \end{array} \right\} & \text{if } z > 0.3 \\ \left(1 - \frac{|z|}{3}\right) & \text{otherwise,} \end{cases}$$

and

$$r_{\text{RELEASE}} = \begin{cases} \left\{ \begin{array}{ll} 0.5\left(1 - \frac{|1-z|}{3}\right) + 0.5 & \text{if } z > z_{t-1}, \\ 0.5\left(1 - \frac{|1-z|}{3}\right) & \text{otherwise} \end{array} \right\} & \text{if } z < 0.3 \\ 1 - \frac{|1-z|}{3} & \text{otherwise.} \end{cases}$$

If the evaluation computed under any of the above conditions is less than zero, it is set equal to 0.

The bead-counting task

The process The process to be controlled is the abacus described in Section 6.3.1.

The controller The network used to implement the controller is shown in Figure B.3. There are three main modules in this network: the level 1 and level 2 controllers and the command filter module.

In it's input layer, the level 1 controller has 8 binary units specifying the command input, 8 binary units providing sensory feedback, and 4 binary units providing feedback of the level's actions at the previous time step. There are 20 backpropagation units in the hidden layer and 4 binary A_{R-P} units are used to generate the control actions. As shown in Figure B.3, the hidden units in this level are connected to the command and sensory input units, while the A_{R-P} units in the output layer of this level are connected to all the input and hidden units. The level 1 task commands are encoded using unit vectors. The encoding of the sensations and the actions are described in Section 6.3.1. The learning rate α was set to 0.01 for all the units in this level. For the A_{R-P} units, λ was set to 0.004.

The level 2 controller has a single binary command input, 11 binary units providing sensory feedback, and 8 more binary units that provide feedback of the level's actions in the previous time step. There are 30 backpropagation units in the hidden layer and the actions of the second level are generated by 8 A_{R-P} units. Again, the hidden units are connected to only the command and sensory input units in this

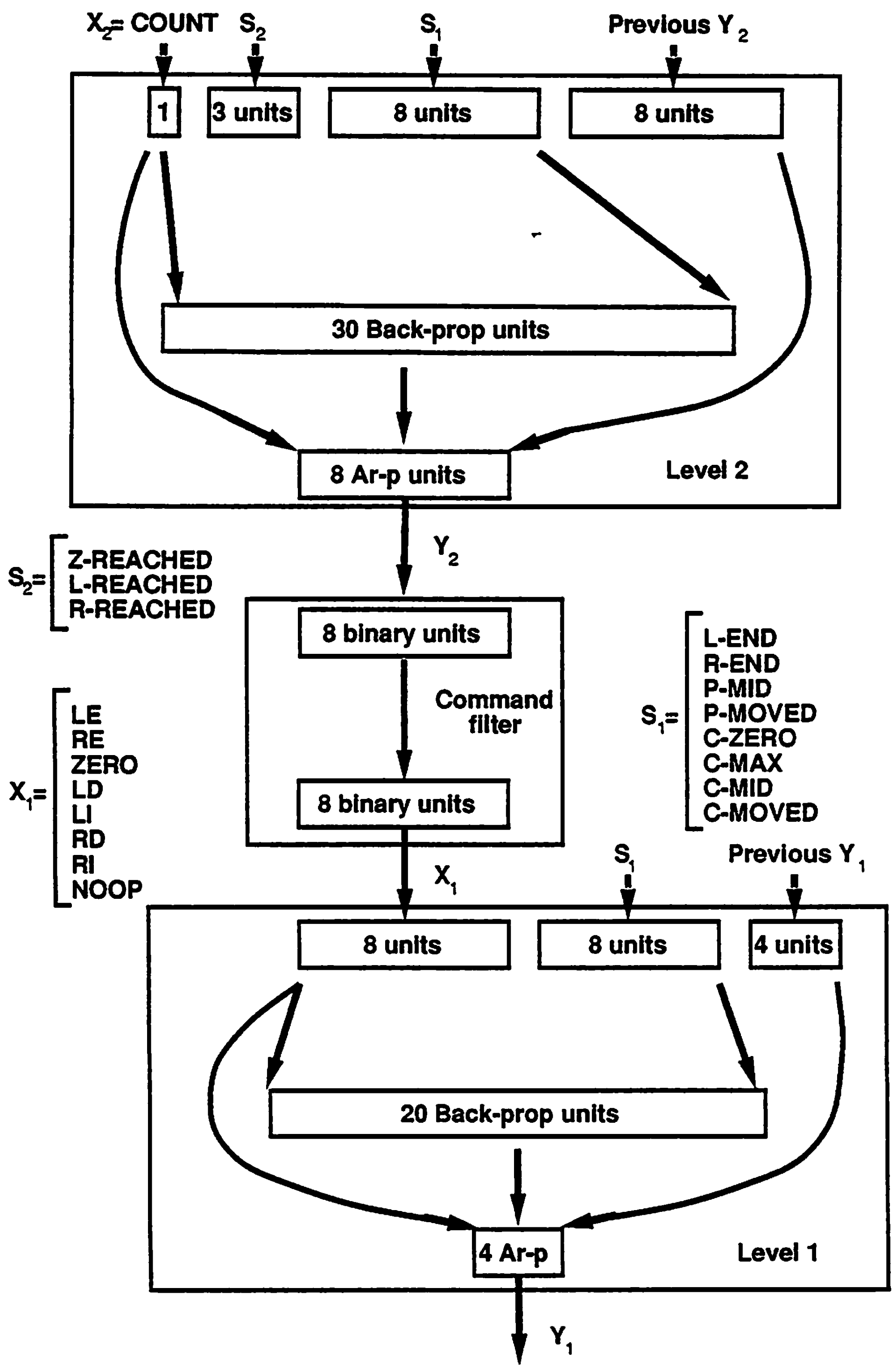


Figure B.3. *The controller network used for the bead counting task.*

level, while the A_{R-P} units in the output layer of this level are connected to both the input and the hidden units of this level (see Figure B.3). Refer to Section 6.3.1 for a description of the sensory inputs to this level. The learning rate α was set to 0.1 for all the units in this level, while λ was set to 0.004 for the A_{R-P} units.

The command filter module was implemented using 8 binary input units which were fully connected to 8 binary output units. This module was trained to function as an autoassociative network using the "Learnmatrix" algorithm described by Hecht-Nielsen [57].

The evaluation The evaluation was computed for each of the level 1 tasks as follows:

LE: $$r = \begin{cases} 1 & \text{if pointer at the left end and counter unchanged.} \\ 0 & \text{otherwise.} \end{cases}$$

RE: $$r = \begin{cases} 1 & \text{if pointer at the right end and counter unchanged.} \\ 0 & \text{otherwise.} \end{cases}$$

ZERO: $$r = \begin{cases} 1 & \text{if counter is zero and pointer has not been moved.} \\ 0 & \text{otherwise.} \end{cases}$$

LD/LI/RD/RI: $$r = \begin{cases} 1 & \text{if task accomplished on the current time step OR} \\ & \text{task accomplished on last time step and NOOP on} \\ & \text{this time step.} \\ 0 & \text{otherwise.} \end{cases}$$

NOOP: $$r = \begin{cases} 1 & \text{if pointer and counter remain the same.} \\ 0 & \text{otherwise.} \end{cases}$$

For the level 2 COUNT task, the evaluation was computed as follows:

$$r = \begin{cases} 0.9 & \text{when either subgoal is accomplished the FIRST time} \\ 1 & \text{if counter is set to number of beads AFTER both subgoals have} \\ & \text{been accomplished.} \\ 0 & \text{otherwise.} \end{cases}$$

The two subgoals for the COUNT task are described in Section 6.3.1.

References

[1] Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9:147–169, 1985.

[2] Albus, J. S. Mechanisms of planning and problem solving in the brain. *Mathematical Biosciences*, 45:247–293, 1979.

[3] Albus, J. S. *Brains, behavior, and Robotics.* BYTE Books, Peterborough, NH, 1981.

[4] Allen, R. B. Adaptive training of connectionist state machines. In *ACM Computer Science Conference*, Louisville, February 1989.

[5] Alspector, J., Allen, R. B., Hu, V., and Satyanarayana, S. Stochastic learning networks and their electronic implementation. In *Proceedings of IEEE Conference on Neural Information Processing Systems—Natural and Synthetic*, Denver, CO, November 1987.

[6] Anderson, C. W. *Learning and Problem Solving with Multilayer Connectionist Systems.* PhD thesis, University of Massachusetts, Amherst, MA, 1986.

[7] Anderson, C. W. Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine*, 9(3):31–37, 1989.

[8] Arimoto, S., Kawamura, S., and Miyazaki, F. Bettering operation of robots by learning. *Journal of Robotic Systems*, 1(2):123–140, 1984.

[9] Asada, H. Teaching and learning of compliance using neural nets: Representation and generation of nonlinear compliance. In *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, pages 1237–1244, 1990.

[10] Atkeson, C. G., and Reinkensmeyer, D. J. Using associative content-addressable memories to control robots. In *IEEE Conference on Decision and Control*, 1988.

[11] Atkinson, R. C., Bower, G. H., and Crothers, E. J. *An Introduction to Mathematical Learning Theory.* Wiley, New York, 1965.

[12] Bachrach, J. R. *Connectionist modeling and control.* PhD thesis, University of Massachusetts, Amherst, MA 01003, May 1991.

[13] Barlow, H. B. Unsupervised learning. *Neural Computation*, 1:295–311, 1989.

[14] Barr, E., and Feigenbaum, E. A., editors. *The Handbook of Artificial Intelligence*, volume 1. William Kaufmann, Inc., Los Altos, CA, 1981.

[15] Barto, A. G. Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiology*, 4:229–256, 1985.

[16] Barto, A. G. Connectionist learning for control: An overview. Technical Report 89-89, University of Massachusetts, Amherst, MA, 1989.

[17] Barto, A. G. Some learning tasks from a control perspective. COINS Technical Report 91-122, University of Massachusetts, Amherst, MA, 1991.

[18] Barto, A. G., and Anandan, P. Pattern recognizing stochastic learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 15:360–375, 1985.

[19] Barto, A. G., and Jordan, M. I. Gradient following without back-propagation in layered networks. In *Proceedings of the IEEE First Annual Conference on Neural Networks*, pages II629–II636, San Diego, CA, 1987.

[20] Barto, A. G., and Singh, S. P. On the computational economics of reinforcement learning. In D. S. Touretsky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton, editors, *Connectionist Models: Proceedings of the 1990 Summer School.* Morgan Kaufmann Publishers, 2929 Campus Drive, Suite 260, San Mateo, CA 94403, 1991.

[21] Barto, A. G., and Sutton, R. S. Simulation of anticipatory responses in classical conditioning by a neuron-like adaptive element. *Behavioural Brain Research*, 4:221–235, 1982.

[22] Barto, A. G., and Sutton, R. S. Neural problem solving. Technical Report 83-03, Department of Computer and Information Science, University of Massachusetts, Amherst, MA, 1983.

[23] Barto, A. G., Sutton, R. S., and Anderson, C. W. Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:835–846, 1983.

[24] Barto, A. G., Sutton, R. S., and Brouwer, P. S. Associative search network: A reinforcement learning associative memory. *IEEE Transactions on Systems, Man, and Cybernetics*, 40:201–211, 1981.

[25] Barto, A. G., Sutton, R. S., and Watkins, C. J. C. H. Learning and sequential decision making. COINS Technical Report 89-95, University of Massachusetts, Amherst, MA, 1989.

[26] Bush, R. R., and Estes, W. K., editors. *Studies in Mathematical Learning Theory.* Stanford University Press, Stanford, CA, 1959.

[27] Caine, M. E., Lozano-Pérez, T., and Seering, W. P. Assembly strategies for chamferless parts. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 472–477, May 1989.

[28] Chandrasekharan, B., and Shen, D. W. C. On expediency and convergence in variable structure automata. *IEEE Transactions on Systems, Man, and Cybernetics*, SSC-4(2):52–60, 1968.

[29] Connell, M. E., and Utgoff, P. E. Learning to control a dynamic physical system. *Comput. Intell.*, 3:330–337, 1987.

[30] Donald, B. R. Robot motion planning with uncertainty in the geometric models of the robot and environment: A formal framework for error detection and recovery. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1588–1593, 1986.

[31] Doob, J. L. *Stochastic Processes.* John Wiley and Sons, New York, 1953.

[32] Draper, C. S., and Li, Y. J. Principles of optimalizing control systems and an application to an internal combustion engine. *ASME Publications*, September 1951.

[33] Duda, R. O., and Hart, P. E. *Pattern Classification and Scene Analysis.* Wiley, New York, 1973.

[34] Dvoretzky, A. On stochastic approximation. In *Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 39–55, Berkeley and Los Angeles, 1956. University of California Press.

[35] Erdmann, M. Using backprojections for fine motion planning with uncertainty. *International Journal of Robotics Research*, 5(1):19–45, 1986.

[36] Ersü, E., and Tolle, H. Learning control structures with neuron-like associative memories. In W. von Seelen, G. Shaw, and U. M. Leinhos, editors, *Organization of Neural Networks.* VCH Verlagsgesellschaft mbH, FRG, 1988.

[37] Fahlman, S. E., and Lebiere, C. The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in neural information processing systems II.* Morgan Kaufman, San Mateo, CA, 1990.

[38] Farley, B. G., and Clark, W. A. Simulation of self-organizing systems by digital computer. *I.R.E Transactions on Information Theory*, 4:76–84, 1954.

[39] Fikes, R. E., Hart, P. E., and Nilsson, N. J. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

[40] Franklin, J. Learning control of a one link robot arm. ECE Technical Report CCS-87-101, Elecreical and Computer Engineering Department, University of Massachusetts, Amherst, Massachusetts, February 1987.

[41] Franklin, J. *Compliance and learning: Control skills for a robot operating in an uncertain world.* PhD thesis, University of Massachusetts, Amherst, MA, February 1988.

[42] Fu, K. S. Learning control systems—Review and outlook. *IEEE Transactions on Automatic Control*, pages 210–221, 1970.

[43] Fu, K. S., and Waltz, M. D. A heuristic approach to reinforcement-learning control systems. *IEEE Transactions on Information Theory*, 9:390–398, 1965.

[44] Gladyshev, E. A. On stochastic approximation. *Theory of Probability and Applications*, 10(2):275–278, 1965.

[45] Goodwin, G. C., and Sin, K. S. *Adaptive Filtering, Prediction, and Control.* Prentice-Hall, Englewood Cliffs, NJ, 1984.

[46] Gordon, S. J. *Automated assembly using feature localization.* PhD thesis, Massachusetts Institute of Technology, MIT AI Laboratory, Cambridge, MA, 1986. Technical Report 932.

[47] Gullapalli, V. A stochastic algorithm for learning real-valued functions via reinforcement feedback. Technical Report 88-91, University of Massachusetts, Amherst, MA, 1988.

[48] Gullapalli, V. A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural Networks*, 3:671–692, 1990.

[49] Gullapalli, V. A comparison of supervised and reinforcement learning methods on a reinforcement learning task. In *Proceedings of the 1991 IEEE International Symposium on Intelligent Control*, pages 394–399, Arlington, Virginia, USA, 13–15 August 1991.

[50] Gullapalli, V. Modeling cortical area 7a using stochastic real-valued (SRV) units. In D. S. Touretsky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton, editors, *Connectionist Models: Proceedings of the 1990 Summer School.* Morgan Kaufmann Publishers, 2929 Campus Drive, Suite 260, San Mateo, CA 94403, 1991.

[51] Gustavson, R. E. A theory for the three-dimensional mating of chamfered cylindrical parts. *Journal of Mechanisms, Transmissions, and Automated Design*, December 1984.

[52] Hampson, S. E. *A Neural Model of Adaptive Behavior.* PhD thesis, University of California, Irvine, CA, 1983.

[53] Handelman, D. A., Lane, S. H., and Gelfand, J. J. Goal-directed encoding of task knowledge for robotic skill acquisition. In *Proceedings of the 1991 IEEE International Symposium on Intelligent Control*, pages 388–393, Arlington, Virginia, USA, 13–15 August 1991.

[54] Hanson, S. J. A stochastic version of the delta rule. *Physica D*, 1989. To appear.

[55] Hanson, S. J., and Pratt, L. Y. Comparing biases for minimal network construction with back-propagation. In D. S. Touretzky, editor, *Advances in neural information processing systems I.* Morgan Kaufman, 1989.

[56] Harth, E., and Tzanakou, E. Alopex: A stochastic method for determining visual receptive fields. *Vision Research*, 14:1475–1482, 1974.

[57] Hecht-Nielsen, R. *Neurocomputing.* Addison-Wesley Publishing Company, 1989.

[58] Hinton, G. E. Connectionist learning procedures. Technical Report CMU-CS-87-115, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa., 1987.

[59] Holland, J. H. Escaping brittleness: The possibility of general-purpose learning algorithms applied to rule-based systems. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach, Volume II.* Morgan Kaufmann, Los Altos, CA, 1986.

[60] Honig, W. K., and Staddon, J. E. R. *Handbook of operant behavior.* Prentice Hall, Englewood Cliffs, NJ, 1977.

[61] Iba, G. A. A heuristic approach to the discovery of macro-operators. *Machine Learning*, 3:285–317, 1989.

[62] Jabri, M., and Flower, B. Weight perturbation: An optimal architecture and learning technique for analog VLSI feedforward and recurrent multi-layer networks. Technical report, University of Sydney, School of Electrical Engineering, 1991.

[63] Jacobs, R. A. *Task decomposition through competition in a modular connectionist architecture.* PhD thesis, University of Massachusetts, Amherst, COINS Department, September 1990.

[64] Jordan, M. I. Supervised learning and systems with excess degrees of freedom. Technical Report 88-27, University of Massachusetts, Amherst, MA, 1988.

[65] Jordan, M. I. Indeterminate motor skill learning problems. In M. Jeannerod, editor, *Attention and Performance, XIII.* The MIT Press, Cambridge, MA, 1990.

[66] Jordan, M. I., and Jacobs, R. A. Learning to control an unstable system with forward modeling. In D. S. Touretzky, editor, *Advances in neural information processing systems II.* Morgan Kaufman, San Mateo, CA, 1990.

[67] Jordan, M. I., and Rosenbaum, D. A. Action. In M. I. Posner, editor, *Foundations of Cognitive Science.* The MIT Press, Cambridge, MA, 1989.

[68] Jordan, M. I., and Rumelhart, D. E. Forward models: Supervised learning with a distal teacher. Center for Cognitive Science Occasional Paper #40, Massachusetts Institute of Technology, Cambridge, MA, 1990.

[69] Kashyap, R. L., Blaydon, C. C., and Fu, K. S. Stochastic approximation. In J. M. Mendel and K. S. Fu, editors, *Adaptive, Learning and Pattern Recognition Systems: Theory and Applications.* Academic Press, New York, 1970.

[70] Kawato, M. Computational schemes and neural network models for formation and control of multijoint arm trajectory. In T. Miller, R. S. Sutton, and P. J. Werbos, editors, *Neural Networks for Control.* The MIT Press, Cambridge, MA, 1990.

[71] Kawato, M., Uno, Y., Isobe, M., and Suzuki, R. Hierarchical neural-network model for voluntary movement with application to robotics. *IEEE Control Systems Magazine*, 8:8–16, 1988.

[72] Kiefer, J., and Wolfowitz, J. Stochastic estimation of the maximum of a regression function. *Ann. Math. Stat.*, 23(3), 1952.

[73] Klopf, A. H. Brain functions and adaptive systems – A heterostatic theory. Technical Report AFCRL-72-0164, Air Force Cambridge Research Laboratories, Bedford, MA, 1972. A summary appears in *Proceedings of the International Conference on Systems, Man, and Cybernetics*, IEEE Systems, Man, and Cybernetics Society, Dallas, TX.

[74] Klopf, A. H. *The Hedonistic Neuron: A Theory of Memory, Learning, and Intelligence.* Hemisphere, Washington, D.C., 1982.

[75] Klopf, A. H., and Gose, E. An evolutionary pattern recognition network. *IEEE Transactions on Systems, Man, and Cybernetics*, 15:247–250, 1969.

[76] Kohonen, T. *Associative Memory: A System Theoretic Approach.* Springer, Berlin, 1977.

[77] Korf, R. E. Macro operators: A weak method for learning. *Artificial Intelligence*, 26:35–77, 1985.

[78] Kuperstein, M. Adaptive visual-motor coordination in multijoint robots using parallel architecture. In *IEEE International Conference on Robotics and Automation*, pages 1595–1602, 1987.

[79] Laird, J. E., Newell, A., and Rosenbloom, P. S. SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64, 1987.

[80] le Cun, Y. Une procedure d'apprentissage pour reseau a sequil assymetrique [A learning procedure for asymmetric threshold network]. *Proceedings of Cognitiva*, 85:599–604, 1985.

[81] le Cun, Y., Denker, J., Solla, S., Howard, R. E., and Jackel, L. D. Optimal brain damage. In D. S. Touretzky, editor, *Advances in neural information processing systems II.* Morgan Kaufman, San Mateo, CA, 1990.

[82] Lee, S., and Kim, M. H. Learning expert systems for robot fine motion control. In H. E. Stephanou, A. Meystal, and J. Y. S. Luh, editors, *Proceedings of the 1988 IEEE International Symposium on Intelligent Control*, pages 534–544, Arlington, Virginia, USA, 1989. IEEE Computer Society Press: Washington.

[83] Lippmann, R. P. An introduction to computing with neural nets. *IEEE ASSP Magazine*, pages 4–22, April 1987.

[84] Ljung, L., and Söderström, T. *Theory and Practice of Recursive Identification.* The MIT Press, Cambridge, MA, 1983.

[85] Lozano-Pérez, T., Mason, M. T., and Taylor, R. H. Automatic synthesis of fine-motion strategies for robots. *The International Journal of Robotics Research*, 3(1):3–24, Spring 1984.

[86] Maciejowski, J. M. *Multivariable Feedback Design.* Addison Wesley, 1989.

[87] Mars, P. Neural nets and robotic control. Technical Report RIPRREP/1000/33/88, School of Engineering and Applied Science, University of Durham, Durham, DH1 3LE, UK, 1988.

[88] Massone, L., and Bizzi, E. A neural network model for limb trajectory formation. Technical report, Department of Brain and Cognitive Sciences, Massachusetts Institute of Technology, Cambridge, MA 02139, 1988.

[89] McClelland, J. L., and Rumelhart, D. E., editors. *Explorations in Parallel Distributed Processing: A handbook of models, programs, and exercises.* Bradford Books/MIT Press, Cambridge, MA, 1988.

[90] Mendel, J. M., and McLaren, R. W. Reinforcement learning control and pattern recognition systems. In J. M. Mendel and K. S. Fu, editors, *Adaptive, Learning and Pattern Recognition Systems: Theory and Applications*, pages 287–318. Academic Press, New York, 1970.

[91] Meystel, A. Intelligent control in robotics. *Journal of Robotic Systems*, 5(4):269–308, 1988.

[92] Michie, D., and Chambers, R. A. BOXES: An experiment in adaptive control. In E. Dale and D. Michie, editors, *Machine Intelligence 2*, pages 137–152. Oliver and Boyd, 1968.

[93] Miller, W. T. Sensor based control of robotic manipulators using a general learning algorithm. *IEEE Journal of Robotics and Automation*, 3:157–165, 1987.

[94] Minsky, M. L. *The Society of Mind.* Simon and Schuster, New York, NY, 1986.

[95] Minsky, M. L. *Theory of neural-analog reinforcement systems and its application to the brain model problem.* PhD thesis, Princeton University, Princeton, NJ, 1954.

[96] Minsky, M. L. Steps toward artificial intelligence. *Proceedings of the Institute of Radio Engineers*, 49:8–30, 1961. Reprinted in E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought.* McGraw-Hill, New York, 406–450, 1963.

[97] Miyata, Y. The learning and planning of actions. ICS Report 8802, Institute for cognitive science, University of California, San Diego, La Jolla, CA 92093, 1988.

[98] Moody, J., and Darken, C. Learning with localized receptive fields. In D. S. Touretsky, G. E. Hinton, and T. J. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School.* Morgan Kaufmann Publishers, 2929 Campus Drive, Suite 260, San Mateo, CA 94403, 1988.

[99] Mozer, M. C., and Smolensky, P. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In D. S. Touretzky, editor, *Advances in neural information processing systems I.* Morgan Kaufman, 1989.

[100] Munro, P. A dual back-propagation scheme for scalar reward learning. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, pages 165–176, Seattle, WA, 1987.

[101] Mussa Ivaldi, F. A., McIntyre, J., and Bizzi, E. Kinematic networks – A distributed model for representing and regularizing motor redundancy. *Biological Cybernetics*, 60:1–16, 1988.

[102] Narendra, K. S. Adaptive control using neural networks. In T. Miller, R. S. Sutton, and P. J. Werbos, editors, *Neural Networks for Control*, chapter 5. The MIT Press, Cambridge, MA, 1990.

[103] Narendra, K. S., and Annaswamy, A. *Stable adaptive systems.* Prentice-Hall, Englewood Cliffs, NJ, 1989.

[104] Narendra, K. S., and Thathachar, M. A. L. *Learning Automata: An Introduction.* Prentice Hall, Englewood Cliffs, New Jersey 07632, 1989.

[105] Nevins, J. L., and Whitney, D. E. Computer controlled assembly. *Science*, 238(2), February 1978.

[106] Newell, A., and Simon, H. *Human Problem Solving.* Prentice-Hall, Englewood Cliffs, NJ, 1972.

[107] Nguyen, D., and Widrow, B. The truck backer-upper: An example of self-learning in neural networks. In T. Miller, R. S. Sutton, and P. J. Werbos, editors, *Neural Networks for Control.* The MIT Press, Cambridge, MA, 1990.

[108] Niranjan, M., and Fallside, F. Neural networks and radial basis functions in classifying static speech patterns. Technical Report CUED/F-INFENG/TR 22, University Engineering Department, Cambridge, CB2 1PZ, England, 1988.

[109] Nowlan, S. J. Gain variation in recurrent error propagation networks. *Complex Systems*, 2:305–320, 1988.

[110] Nowlan, S. J., and Hinton, G. E. Evaluation of adaptive mixtures of competing experts. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in neural information processing systems 3.* Morgan Kaufman, San Mateo, CA, 1991.

[111] Parker, D. B. Learning logic. Technical Report TR-47, Massachusetts Institute of Technology, 1985.

[112] Quinlan, J. R. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[113] Robbins, H., and Monro, S. A stochastic approximation method. *Ann. Math. Stat.*, 22(1):400–407, 1951.

[114] Rosenblatt, F. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Spartan Books, 6411 Chillum Place N.W., Washington, D.C., 1961.

[115] Rumelhart, D. E. Lecture at the 1988 connectionist models summer school, 1989.

[116] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol.1: Foundations.* Bradford Books/MIT Press, Cambridge, MA, 1986.

[117] Rumelhart, D. E., and McClelland, J. L., editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol.1: Foundations, Vol. 2: Psychological and Biological models.* Bradford Books/MIT Press, Cambridge, MA, 1986.

[118] Rumelhart, D. E., and Zipser, D. Feature discovery by competitive learning. *Cognitive Science*, 9:75–112, 1985.

[119] Samuel, A. L. Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, pages 210–229, 1959. Reprinted in E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, McGraw-Hill, New York, 1963.

[120] Samuel, A. L. Some studies in machine learning using the game of checkers. II—Recent progress. *IBM Journal on Research and Development*, pages 601–617, November 1967.

[121] Saridis, G. N. *Self-organizing Control of Stochastic Systems.* Marcel Dekker, Inc., New York, 1977.

[122] Schmetterer, L. Stochastic approximation. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 587–609, Berkeley and Los Angeles, 1961. University of California Press.

[123] Selfridge, O. Pandemonium: A paradigm for learning. In *Proceedings of the Symposium on the Mechanisation of Thought Processes*, Teddington, England: National Physical Laboratory, H.M. Stationary Office, London, 2 vols, 1959.

[124] Selfridge, O. Learning to count. How a computer might do it. In preparation, 1979.

[125] Selfridge, O., Sutton, R. S., and Barto, A. G. Training and tracking in robotics. In *Proceedings of the Ninth International Joint Conference of Artificial Intelligence*, Los Angeles, CA, Aug. 1985.

[126] Simons, J., Brussel, H. V., Schutter, J. D., and Verhaert, J. A self-learning automaton with variable resolution for high precision assembly by industrial robots. *IEEE Transactions on Automatic Control*, 27(5):1109–1113, October 1982.

[127] Skinner, B. F. *The Behavior of Organisms: An experimental analysis.* D. Appleton Century, New York, 1938.

[128] Skinner, B. F. "Superstition" in the pigeon. *Journal of Experimental Psychology*, 38:168–172, 1948.

[129] Skinner, B. F. *Science and Human Behavior.* Macmillan, New York, 1953.

[130] Skolic, M. E. Adaptive model for decision making. *Pattern Recognition*, 15:485–493, 1982.

[131] Srinivasan, V., Barto, A. G., and Ydstie, B. E. Pattern recognition and feedback via parallel distributed processing. In *Annual Meeting of the AIChE*, Washington D. C., November 1988.

[132] Staddon, J. E. R. *Adaptive behavior and learning.* Cambridge University Press, 1983.

[133] Stafford, R. A. Multi-layer learning networks. In J. E. Garvey, editor, *Symposium on self-organizing systems.* Office of Naval Research, 1963.

[134] Stafford, R. A. A learning network model. In M. Maxfield, A. Callahan, and L. Fogel, editors, *Biophysics and cybernetic systems.* Spartan Books Inc., Washington, DC, 1965.

[135] Sutton, R. S. Ballistic bug, June 1982. Unpublished working paper.

[136] Sutton, R. S. *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, University of Massachusetts, Amherst, MA, 1984.

[137] Sutton, R. S. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[138] Sutton, R. S., and Barto, A. G. An adaptive network that constructs and uses an internal model of its world. *Cognition and Brain Theory*, 3:217–246, 1981.

[139] Tangwongsan, S., and Fu, K. S. An application of learning to robot planning. *International Journal of Computer and Information Sciences*, 8(4), 1979.

[140] Tsetlin, M. L. *Automaton Theory and Modeling of Biological Systems.* Academic Press, New York, 1973.

[141] Watkins, C. J. C. H. *Learning from delayed rewards.* PhD thesis, Cambridge University, Cambridge, England, 1989.

[142] Weiland, A. P., and Leighton, R. R. Shaping schedules as a method for accelerating learning. In *International Neural Network Society Meeting*, 1988.

[143] Werbos, P. J. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University, 1974.

[144] Werbos, P. J. Backpropagation and neurocontrol: A review and prospectus. In *Proceedings of the 1989 International Joint Conference on Neural Networks*, Washington, D.C., June 1989.

[145] Werbos, P. J. A menu of designs for reinforcement learning over time. In T. Miller, R. S. Sutton, and P. J. Werbos, editors, *Neural Networks for Control*, chapter 3. The MIT Press, Cambridge, MA, 1990.

[146] Whitney, D. E. Quasi-static assembly of compliantly supported rigid parts. *Journal of Dynamic Systems, Measurement, and Control*, 104, March 1982. Also in *Robot Motion: Planning and Control*, (Brady, M., et al. eds.), MIT Press, Cambridge, MA, 1982.

[147] Widrow, B. Generalization and information storage in networks of adaline "neurons". In M. Yovits, G. Jacobi, and G. Goldstein, editors, *Self-organizing systems.* Spartan Books, 1962.

[148] Widrow, B., and Hoff, M. E. Adaptive switching circuits. In *1960 WESCON Convention Record Part IV*, pages 96–104, 1960.

[149] Widrow, B., McCool, J., and Medoff, B. Adaptive control by inverse modeling. In *Twelfth Asilomar Conference on Circuits, Systems, and Computers*, 1978.

[150] Widrow, B., and Smith, F. W. Pattern-recognizing control systems. In *Computer and Information Sciences (COINS) Proceedings*, Washington, D.C., 1964. Spartan.

[151] Wieland, A. P. Evolving controls for unstable systems. In D. S. Touretsky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton, editors, *Connectionist Models: Proceedings of the 1990 Summer School.* Morgan Kaufmann Publishers, 2929 Campus Drive, Suite 260, San Mateo, CA 94403, 1991.

[152] Williams, R. J. Reinforcement learning in connectionist networks: A mathematical analysis. Technical Report ICS 8605, Institute for Cognitive Science, University of California at San Diego, La Jolla, CA, 1986.

[153] Williams, R. J. Reinforcement-learning connectionist systems. Technical Report NU-CCS-87-3, College of Computer Science, Northeastern University, 360 Huntington Avenue, Boston, MA, 1987.

[154] Williams, R. J. On the use of backpropagation in associative reinforcement learning. In *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, 1988.

[155] Wolfowitz, J. On the stochastic approximation method of Robbins and Monro. *Ann. Math. Stat.*, 25:457–461, 1952.

[156] Ydstie, B. E. Forecasting and control using adaptive connectionist networks. *Computers Chem. Enging.*, 14(4/5):583–599, 1990.