TOWARDS PLANNING: INCREMENTAL INVESTIGATIONS INTO ADAPTIVE ROBOT CONTROL

Lisa A. Meeden

Submitted to the faculty of the Graduate School
in partial fulfillment of the requirements
for the degree
Doctor of Philosophy
in the Department of Computer Science
Indiana University

August 1994

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements of the degree of Doctor of Philosophy.

Doctoral Committee

Michael Gasser, Ph.D.
(Principal Advisor)

David Leake, Ph.D.

Robert F. Port, Ph.D.

Paul Purdom, Ph.D.

Linda Smith, Ph.D.

Copyright © 1994
Lisa A. Meeden
ALL RIGHTS RESERVED

Acknowledgements

My advisor, Mike Gasser, has been a steady influence throughout my graduate career. During our frequent meetings, Mike always seemed to offer just the right amount of direction. He allowed me to work independently, but quickly provided encouragement and advice when I got off track. I would also like to express my thanks to the rest of my committee, David Leake, Bob Port, Paul Purdom, and Linda Smith.

Doug Blank and Gary McGraw provided a crucial contribution to this work: the robot which I call carbot. While attending the Artificial Life III conference, they entered a robot building contest and the forerunner to carbot was the result. This initial robot had an illustrious start, winning both the "Most Musical" and "Most Obnoxious" categories for its rendition of "Rocky's Theme" at the conclusion of the maze. When they returned from the conference they wanted to continue tinkering with robots and began building what would become carbot. The three of us wrote a paper together about carbot, but I eventually managed to commandeer carbot for my own purposes.

Doug and Gary contributed in other important ways as well. Doug has patiently listened to my ideas about planning and discussions with him frequently led to key insights into the problem. Gary allowed me to take over his office while he was away in Italy for a year. I thank him, Helga Keller, Doug Hofstadter, and the Center for Research on Concepts and Cognition for providing a quiet, comfortable place to work. I don't think I could have finished without it.

I would like to thank Jonathan Mills for inviting me to attend the workshop "On learning and adaptation in robots and situated agents" at the Santa Fe Institute. This proved to be a very stimulating gathering that led me to explore new areas in my research.

In my first years of graduate work, Jim Marshall and I managed to attend exactly the same set of classes, and he proved to be a consummate study partner. Jim doesn't give up on a problem until he completely understands it. This trait caused me to delve more deeply into computer science than I would have done on my own, and I thank him for that.

Whenever I was stuck on something, someone in the Artificial Intelligence lab, which included Fred Cummins, Susan Fox, Devin McAuley, John Nienart, Cathy Rogers, and Raja Sooriamurthi, could usually help me figure out why. John Nienart tirelessly answered any Unix, C, and Scheme questions I managed to come up with. I greatly appreciated his hacking skills.

I am also glad I got to share my time in the Computer Science department with Pete Beckman, Tim Bridges, Anand Deshpande, Eric Jeschke, and Jon Rossie.

The promise of ultimate frisbee at the end of the day was the best incentive for keeping me at my desk. Thanks to all the members of Calamity Jane and the HoosierMama?s for letting me be part of the team, but especially to Amy and Dottie. It's hard to imagine playing ultimate without the two of you.

Finally, thanks to Andy for taking care of things during the final months of writing. He left me no good excuses for avoiding work.

Abstract

Traditional models of planning have adopted a top-down perspective by focusing on the deliberative, conscious qualities of planning at the expense of having a system that is connected to the world through its perceptions. My thesis takes the opposing, bottom-up perspective that being firmly situated in the world is the crucial starting point to understanding planning. The central hypothesis of this thesis is that the ability to plan developed from the more primitive capacity of reactive control.

Neural networks offer the most promising mechanism for investigating robot control and planning because connectionist methodology allows the task demands rather than the designer's biases to be the primary force in shaping a system's development. Input can come directly from the sensors and output can feed directly into the actuators creating a close coupling of perception and action. This interplay between sensing and acting fosters a dynamic interaction between the controller and its environment that is crucial to producing reactive behavior. Because adaptation is fundamental to the connectionist paradigm, the designer need not posit what form the internal knowledge will take or what specific function it will serve. Instead, based on the training task, the system will construct its own internal representations built directly from the sensor readings to achieve the desired control behavior. Once the system has reached an adequate level of performance at the task, its method can be dissected and a high-level understanding of its control principles can be determined.

This thesis takes an incremental approach towards understanding planning. In the initial phase, several ways of representing goals are explored using a simulated robot in a one-dimensional environment. Next the model is extended to accommodate an actual physical robot and two reinforcement learning methods for adapting the network controllers are compared: a gradient descent algorithm and a genetic algorithm. Finally, the model's behavior and representations are analyzed to reveal that it contains the potential building blocks necessary for planning. By actively restricting the extent of our presuppositions about planning, we may be able to develop truly autonomous robots with radically different forms of control and planning.

Contents

List of Tables

List of Figures

1

Reconsidering planning

As I began working on this thesis I did an informal survey on people's definitions of plans. Although everyone seemed to have strong intuitions about what constitutes a plan, they were frequently unwilling to actually commit to a particular answer. The responses were similar to what a famous judge once said about obscenity, "I can't define it, but I know it when I see it." When pressed further, people would describe plans as methods, procedures, programs of action, or sets of explicit instructions. These definitions suggest that we believe plans to be static, self-contained entities. To see plans as static leads one to view the world as relatively stationary as well. To see plans as self-contained implies that they are independent of any particular domain or actor, like a fortran program that can be run on any computer. The actual process of planning is seen as the ability to contemplate future outcomes prior to acting.

These intuitions about the characteristics of planning have profoundly influenced the course of planning research. If the world is basically predictable, if plans are truly self-contained, and if the process of planning is like deliberation, then perception and action should be peripheral issues and the core of the matter is in discovering how such fixed, independent plans are reasoned out.

But if these intuitions are wrong, then the traditional approach to planning must be completely reconsidered. I have a very different set of intuitions about planning. I believe that plans are context-dependent, dynamic entities, which are affected by the moment-to-moment changes in the environment. I view the act of planning more as informed improvisation than as deliberation. Because our environment is continually in flux, an explicit program of action produced with careful forethought will quickly become obsolete. Instead, the process of planning should be brief and the results sketchy—any detailed predictions will most likely be wrong.

The reconsideration of traditional planning has already begun under the new research approach termed *situated activity* or *reactive planning* (Agre, 1988; Chapman, 1991; Firby, 1989). The adjectives *situated* and *reactive* are illustrative of the tenets of this new view. Instead of building a disembodied planner, an autonomous agent is constructed and situated in realistic environments. The agent must constantly react to the dynamic world while still pursuing its long-term goals. Clearly an agent inhabiting an unpredictable world is quite dependent on perceptual and motor skills for success.

This chapter begins with a review of the larger set of assumptions, or *worldview*, that encompasses the deliberative intuitions about plans. I will apply this worldview's conception of planning to the problem of controlling physical robots to illustrate a serious flaw in its fundamental assumptions. I will argue, as other proponents of reactive planning have done, that it is more fruitful for the purpose of robot control to work within an alternative worldview where perception and action are the central concerns.

The alternative worldview to which I subscribe encompasses the concerns of reactive planning as well as those of connectionism. Like connectionists, I believe that learning should be fundamental to the process of cognition. Within this alternative worldview that combines situatedness and learning, plans can be seen as dynamic, context-dependent entities and the adaptation of planning behavior can be considered. Based on this alternative view, I will offer an incremental approach to the development of plans in robots.

1.1 The deliberative processing worldview

> A *worldview* is a largely unarticulated system of vocabulary, methods, and values shared by a research community. (Agre, 1993, p 62)

> A worldview helps its owner interpret observations and guide further studies. It is neither right or wrong. Instead, it has to be judged on how useful it is to the pursuit of science and to a shared understanding of the phenomena. (Norman, 1993b, p 5)

As Agre and Norman suggest, a worldview provides a filter through which observations are made. This filter enhances the significance of some phenomena while diminishing the importance of others, narrowing the sphere of inquiry to a particular focus. Deliberative processing has been the dominant worldview for both Cognitive Science and Artificial Intelligence. Others have referred to this view as *rationalism* to indicate that cognition is explained in terms of logical structures (Pfeifer and Verschure, 1992a). The deliberative or rationalist view has led researchers to concentrate on abstract tasks that more easily lend themselves to a high-level, symbolic formulation, especially problems related to what have been been considered the uniquely human aspects of intelligence—consciousness, language, and planning. Therefore, the deliberative processing view tends to emphasize the crucial differences between humans and other animals while placing much less importance on their common evolutionary paths.

The deliberative processing view assumes that reasoning has largely supplanted reactive behavior commonly found in animals. Yet no explanation is offered for how this conversion from reaction to reasoning might have occurred. Typically models colored by this worldview begin their inquiry from the top, positing highly structured symbolic representations of knowledge that are manipulated according to rules of logic to produce behavior.

One instantiation of the deliberative processing view is Soar—a suggested architecture for general intelligence (Laird et al., 1987). I will focus on Soar because it is representative of the deliberative style of processing and it has been successfully applied to the widest range of problems to date.

Soar embodies two hypotheses fundamental to the deliberative processing worldview. First and foremost is the belief that a general intelligence must be realized with a symbolic system. This is called the *physical symbol system hypothesis* (Newell and Simon, 1972). A second assumption is that problem spaces are the basic organizational unit of all goal-directed behavior. A *problem space* consists of a set of possible states in a particular domain and a set of primitive operators that transform one state into another. Figure 1.1, adapted from (Laird et al., 1987, p 6), depicts the problem space for a particular problem called the eight puzzle.

Planning for a task in Soar is seen as searching through a problem space for an appropriate sequence of operators that will transform the initial state to a desired goal state. Only then is the plan executed. Elaine Rich notes that "for problems such as the eight-puzzle, the distinction between planning and doing is unimportant" (Rich, 1983, p 249). This is because the world of the eight-puzzle is stationary. The computer does not physically push any of the tiles around or interact with the real world in any way.

For the deliberative view, as illustrated by Soar, processing is functionally decomposed into three main modules: perception, reasoning, and execution (see Figure 1.2). First, the task is perceived according to the initial and goal states, then a solution is fully reasoned out using a search through the appropriate problem space, and finally the computed actions are executed. The emphasis in the deliberative view is firmly

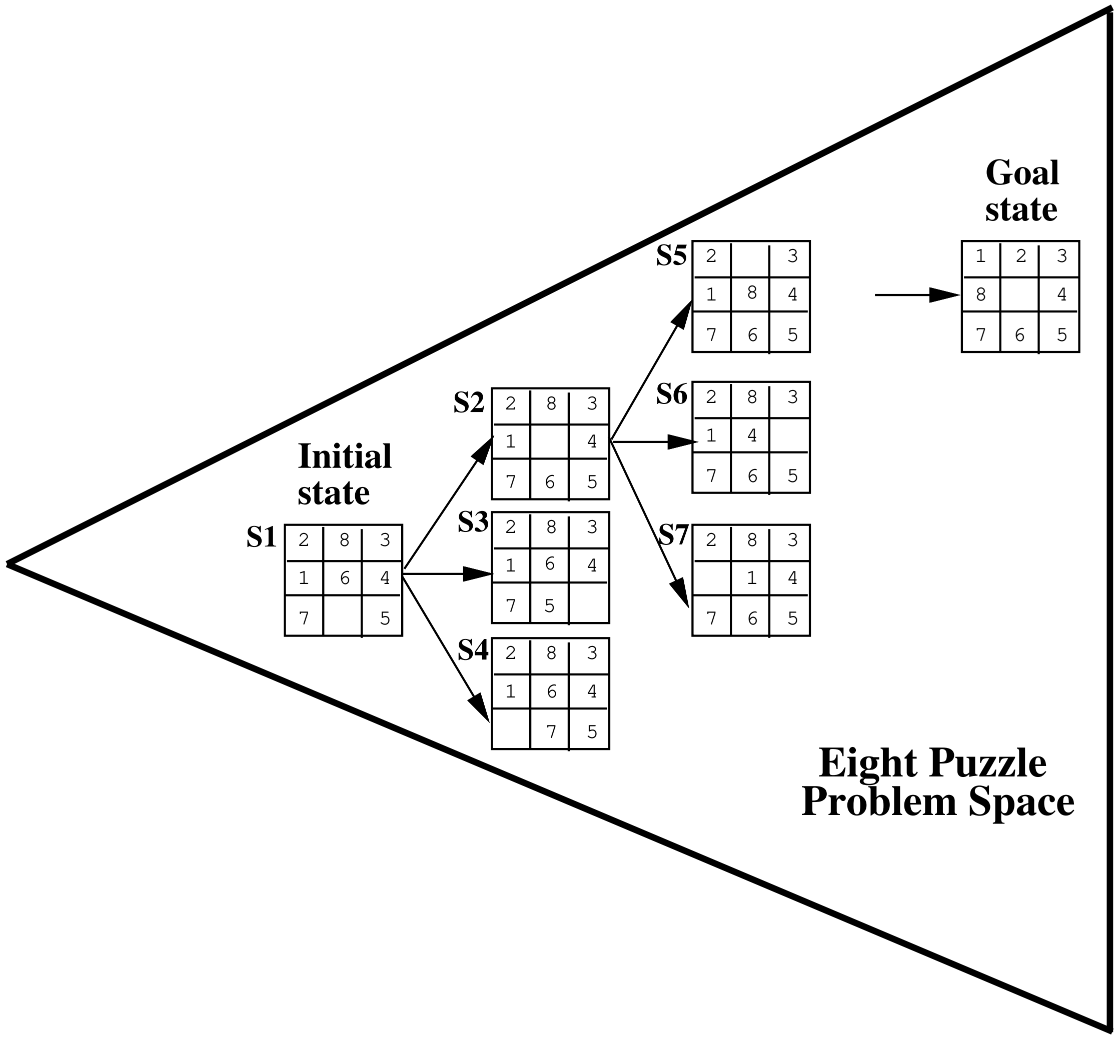

Figure 1.1: The structure of a problem space search for the eight puzzle. There are eight numbered tiles and one blank location configured on a 3x3 board. The operators are tile movements into the blank location (either `left`, `right`, `up` or `down`). State S5 will eventually lead to the goal.

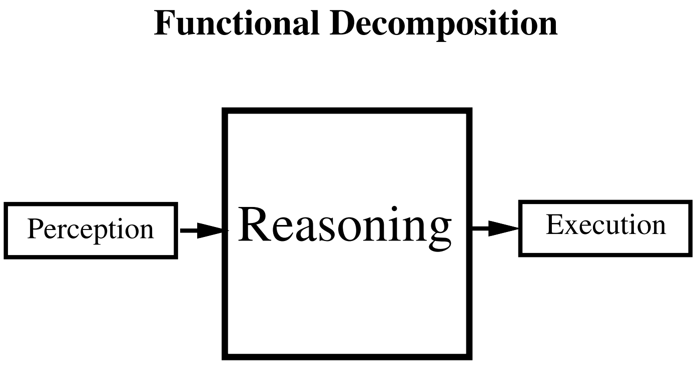

Figure 1.2: The deliberative worldview's decomposition of behavior. Typically, deliberative systems are not connected to any external environment, but operate on internal models. Therefore no input from sensors into Perception or output from Execution to actuators is shown. The Reasoning module encompasses the primary computational effort.

on the reasoning phase of this process. Perception and action are seen as auxiliary functions.

Soar has impressively mimicked human problem solving abilities for numerous tasks, including the eight puzzle, tic-tac-toe, towers of Hanoi, missionaries and cannibals, algebraic equations, logical syllogisms, and blocks world problems (Norman, 1993a). What do these diverse problem domains have in common?

Because deliberative processing models like Soar plan by searching through problem spaces, they have been quite successful when applied to problems with the following features:

- The domain can be described with discrete states.
- Tasks in the domain can be defined by an initial state and a goal state.
- Given the current state and an operator, the next state can be uniquely determined.

Consider the eight puzzle again according to these features. Each state is a board configuration and each tile must be at a discrete location on the board; puzzle tasks are naturally described in terms of beginning and ending states; and eight puzzle operators produce completely predictable results. The eight puzzle domain does indeed fit the stated criteria, thus making it a perfect candidate for deliberative processing models.

However, the usefulness of a worldview depends on the extent to which it can naturally encompass the majority of important phenomena. Given a clearer picture of where the deliberative processing view succeeds, we can now see where it falls short. The deliberative view emphasizes reasoning over perception and action. If we take perceptual and motor skills seriously, by building an autonomous robot that is situated in the real world, how will Soar and the deliberative worldview fare? In summarizing Soar's accomplishments, even Soar's proponents have noted that this is one area which has received little attention:

> ... all of these demonstrations are essentially internal—both planning and execution occur completely within the scope of the system. Thus they do not involve direct execution in a real external environment and they safely ignore many of the issues inherent to such environments. (Laird and Rosenbloom, 1993)

Let's posit a very simple robot and consider in detail how a model based on the deliberative processing worldview would be applied to the problem of controlling this robot. Suppose this robot is limited to moving within a small, empty room which has a light in one corner. The robot is equipped with two light sensors, on its left and right sides, and two short-range sonar sensors at its front and back. Each light sensor returns continuous values between 0.0 and 1.0, where higher values indicate more light. The sonar sensors return the discrete values zero and one. A zero indicates that there is no obstacle detected within sonar range, and a one indicates the robot is within range of an obstacle. The robot can execute four actions: `forward`, `forward-left`, `forward-right`, and `backward`.

Let's choose a very simple navigation task for our robot: from any initial position in the room it must go to the light without contacting any of the walls. We will define success at reaching the light when the sum of the light readings are greater than some minimum value. To attempt a solution within the deliberative worldview we must convert this robot domain and task into a problem space.

First we need to define the states of the problem space. The deliberative worldview defines the state of a system from a global perspective, which for the simple robot under consideration would include its current position and heading. However, an autonomous robot has only its local sensor readings and must deduce from these the global situation. Thus from the robot's perspective, a state for this domain consists only of a four-item list of the sensor readings. It could take the form: $(left-light, right-light, front-sonar, back-sonar)$ with one instantiation being: (0.34, 0.26, 0, 1). This state indicates that there is slightly more light coming from the left light sensor than the right and that some obstacle is sensed behind the robot.

There's a problem though: these sensor states contain continuous values. To perform search-based planning we will need a limited set of states. So there must be some way to classify these continuous states into a discrete set. This can be accomplished by determining ranges in the continuous values. For the navigate-to-light task, we might want to monitor light readings L where $L <= 0.1$, $0.1 < L <= 0.8$, and $L > 0.8$. But by designating such ranges for the light readings some information will be lost. In other words, with this set of ranges a robot controller could not make a decision based on whether $L < 0.5$. The designer must ensure that the system has access to the essential distinctions in the continuous parameters.

Next we must define the task in terms of an initial state and a goal state. The initial state will simply be the discretized sensor readings obtained at the initial position of the robot, such as $(left - light < 0.1, right - light < 0.1, 0, 1)$. The goal state should describe the desired results. Suppose that the sum of the light readings must be greater than 1.6 for success. Then the goal state would be: $(left - light > 0.8, right - light > 0.8, 0, 0)$. But it would also be possible to accomplish the goal with one light reading in the top of the middle range and the other light reading in the top of the highest range. Interestingly there isn't necessarily a single goal state for the task. These additional goal states will certainly complicate the state space search, but let's assume that we can modify the search algorithm to accomodate multiple goal states.

The final hurdle to describing the problem space for our robot is to define a function that maps a current state and action into the next state. Here we face a major hurdle. Because we are using a real robot with actual sensors and motors, there will be noise. The sonar sensors will misfire, indicating an upcoming obstacle where there is none, or failing to indicate a wall in the near vicinity. The light sensor readings will also contain significant noise as a result of picking up ambient light in the room. Even more problematic though, the results of actions will be noisy as well. Performing the same movement will produce different results based on the friction encountered in the environment and other mechanical factors. Neither the new heading nor distance traveled can be accurately predicted. The subsequent light readings depend on the robot's heading and position relative to the light. Similarly, the subsequent sonar readings depend on the robot's position relative to the walls.

Due to the large amounts of noise in both the perception and action processes, the next state cannot be uniquely determined given only the current state and the action to be taken. The inability to determine how the state of a system will change after executing an action is referred to as the *frame problem*. An action planning system based on the deliberative worldview must be able to precisely establish the results of actions without actually executing them. Suppose the robot's current state is $(left-light > 0.8, 0.1 < right-light <= 0.8, 0, 0)$ and that it takes a `forward-left` action, how will these values change? We can guess that moving in the left direction may decrease the left light reading and increase the right reading, but by how much? Will the changes in the light be enough to drop the left light reading to the lower range and push the right light reading to the higher range? Will the robot be within sonar range of a wall? In the real world the outcome of an action cannot be reliably predicted prior to executing it.

This difficulty cannot be side-stepped because perfect sensors and perfect performance will never be obtained. If the global state of the robot (its position and heading) could somehow be provided to the system then the noisiness inherent in perceiving and acting could be safely ignored. However, it is unreasonable to assume that an autonomous agent could ever have exact knowledge of its global state. For human beings this might equate to constantly knowing our precise location on the earth in terms of longitude and latitude. The only viable method for discovering the robot's next global state is to actually execute the action in the real world, determine the local state by re-reading the sensors, and make inferences based on these values.

Soar and other deliberative systems depend on internal models of the problem space on which to perform their planning search. As we've just seen, when the problem space is extended to the real world, managing this internal model's accuracy becomes extremely difficult. If we have to execute each action to determine its outcome, we cannot use *extended* search to plan because we can no longer determine a complete and reliable path before acting. The outcome of each projected action is noisy, and a long series of actions based on noisy predictions will be inaccurate. However, search may still be a viable method if its depth is very limited.

Despite these fundamental problems with the deliberative processing approach to planning in the real world, Soar has been successfully applied to two real robot tasks (Laird et al., 1989; Laird and Rosenbloom, 1993). Soar's solution to this internal truth-maintenance problem has been to abstract away from the primitive actions, operating the search at a much higher level where accurate predictions about outcomes are possible.

Hero-Soar controls a mobile robot equipped with an arm and sonar sensors. Hero-Soar has the task of collecting cups and depositing them in the trash. As discussed previously, extended search at the level of the basic motor commands—to position the arm, manipulate the gripper, and move the robot—would be useless due to the various types of noise inherent in the interaction between the system and the environment. Instead, Hero-Soar is given a higher level of operators, such as `search-for-object`, `orient-toward-object`, `approach-object`, `pickup-cup`, and `drop-cup` on which to do extended search. Executing each of these abstract operators involves a combination of many basic motor commands. Ensuring that the appropriate primitive commands are actually invoked is handled at run-time.

On the surface this is another impressive achievement for the deliberative worldview, but in evaluating this result we should be concerned with how easily it was obtained. Do the extensions needed to accommodate real environments fit naturally into the existing framework? To a certain extent, this is a question of aesthetics, yet there is a deeper practical issue as well. How has the fundamental problem of noise been dealt with?

Let's examine the implementation of Soar's robotics solution a little more closely. Hero-Soar was programmed to pick up cylindrical cups. When it encounters a cone-shaped cup for the first time, it tries to use the original technique, but the new cups slip from its gripper. When something unexpected happens during run-time, such as dropping a cup, the system must have some way to avoid infinitely retrying the same incorrect actions. It must recognize that it needs to learn something new. In this case it needs to learn to distinguish cone-shaped cups from cylindrical cups via its sensor readings and it must also learn a new pick-up technique for cone-shaped cups.

Adaptation in Soar is based on a simple experience-based learning method called *chunking*. Chunking combines existing knowledge with results from a given problem experience and converts it through generalization into new more efficient knowledge available for future problem solving. Soar's knowledge is encoded as a production system and each chunk is a new if-then rule that is added to the production memory.

When an error arises, as when Hero-Soar is unable to pick up the cup, Soar's general strategy is to assume that the internal knowledge could be faulty and to force the system to reconsider each decision. This reconsideration is implemented by "creating a dummy operator and making preferences that make it both better and worse than the other operators" (Laird et al., 1989, p 421). This dummy operator will now be valid in every situation, and because it is better and worse than all other decisions an impasse is created. At this point, human intervention is employed to guide Soar to attend to the features of the problem space causing the error. So rather than allowing the system to experiment in the environment and discover the relevant characteristics on its own, the human advisor directs the learning. Laird acknowledges that this is a "brute-force technique to learn new features" and "although it may not be considered the most elegant or complex machine learning technique, it allows the human to easily correct the system" (Laird et al., 1989, p 422).

A system that depends on human intervention cannot be truly autonomous, but Laird argues that the human element could easily be eliminated by allowing the system to engage in experimentation. No specifics are offered for how this experimentation would take place except that the system could guess at relevant features. The problem though is that Soar's core ability to generalize is dependent on having well-defined, clear-cut features to manipulate:

> If the representation is organized so that aspects that are relevant are factored cleanly from the parts that are not (i.e. are noise) then chunking can learn highly general concepts. Factoring implies both that the aspects are encoded as distinct attributes and that the operators are sensitive only to the relevant attributes and not to the irrelevant attributes. (Laird et al., 1987, p 56)

Chunking is a powerful method for efficiently recombining *consistent* and *existing* knowledge. Allowing Soar to actually extend its sphere of knowledge beyond the boundary of the original relevant features is a major new step. Experimentation will infuse Soar's memory with noisy, irrelevant data, and it is not clear that Soar will remain effective in these conditions. Thus, I do not believe it will be a simple matter to remove the human element.

One of the most difficult aspects of the real world is the prevalence of noise. By a close interaction with the environment this noise can be filtered out and important regularities can be discovered. But the planning in Hero-Soar operates at an abstract level, separate from either perception or action. Thus it is always shielded from the real world. How can a system designed to operate as though there were no noise be simply adjusted so that it can now discover the relevant features amid the noise? Hero-Soar misses the mark on both aesthetic and practical levels.

As the Hero-Soar example shows, even when the deliberative worldview is applied to physical robots in real environments, perception and action are still being sidestepped in deference to high-level guidance. There are fundamental problems with accommodating realistic perception and action within deliberative processing models like Soar. As one would expect from the underlying assumptions of the deliberative view, its models are best suited for producing behavior at the symbolic level. The most impressive achievements encompassed by the deliberative view have been on reasoning tasks where perception and motor control are ignored.

In the next section, I will describe an alternative to the deliberative processing view that addresses the concerns raised here and which borrows aspects from both the *situated action* and *connectionist* worldviews.

1.2 An alternative worldview: Adaptation from limited designer bias

A deep-seated criticism of the deliberative worldview, made by proponents of connectionism, is that to assume that reasoning is a modular process essentially separate from perception and action as previously shown in Figure 1.2 leads to the *symbol grounding problem* (Harnad, 1990; Gasser, 1993; Pfeifer and Verschure, 1992a; Smolensky, 1988). The expectation of the deliberative view has been that the output from some perceptual module can easily be funneled into a reasoning module to produce the right sort of symbol structures to perform search-based reasoning. Yet as AI has begun to move from toy domains to implementing actual physical robots, this modular formulation no longer seems practically feasible. How can meaning ever get appropriately attached to the internal symbols? Harnad describes the problem as follows:

> How can the semantic interpretation of a formal symbol system be made *intrinsic* to the system rather than parasitic on the meanings in our heads? How can the meaning of the meaningless symbol tokens, manipulated solely on the bases of their (arbitrary) shape, be grounded in anything but other meaningless symbols? (Harnad, 1990, p 335)

One solution to this problem is to construct symbols from the ground up, basing them directly on nonsymbolic, perceptual features. In this view it is nonsensical to posit a reasoning level where free-floating symbols are disconnected from the perceptual level because ungrounded symbols are by definition meaningless.

Another criticism of the deliberative processing approach, made by the proponents of the situated action worldview, is that perception and action are more central to people and animal's daily activities than in-the-head problem solving, and therefore reasoning should not be the primary focus of cognitive study (Brooks, 1991; Suchman, 1987)). Instead they suggest that much of the structured behavior we assume must be the result of high-level planning may in fact emerge naturally from an agent's improvisational interaction with the environment.

Agre's quote in the opening of the chapter noted that the assumptions that define a worldview are often not clearly articulated. One far-reaching yet implicit belief within the deliberative view is that the organized nature of behavior is directly due to planning. This belief biases our observations of other agents' behavior:

> Anticipation and planning are of course impossible to observe directly in another person or animal, but indications of their likelihood are often

observable. (Griffin, 1984, p 37)

Griffin argues that complex, structured behavior suggests that planning is taking place. As an example of what he means by structured behavior, he offers a description of the behavior of an assassin bug that lives in the tropical rain forests. This bug uses a very effective strategy for capturing termites. The assassin bug glues small bits of the outer layers of a termite nest to its body. Apparently these bits of nest provide camouflage. Then the bug reaches into the nest opening and captures a termite and kills it by sucking out all the internal organs, leaving only an exoskeleton. The assassin bug then pushes the empty exoskeleton back into the opening and jiggles it gently. Another termite seizes the corpse as part of a normal behavior pattern of devouring dead siblings. The assassin bug pulls this second termite out and eats it as well. In one case an assassin bug was observed to eat 31 termites in this manner (Griffin, 1984, p 123).

A key contribution of the situated action view is that it questions this implicit belief in the centrality of planning. If insects like the assassin bug can produce highly structured behavior, then structured behavior may be the result of much simpler mechanisms than planning. One possibility, to which I subscribe, is that planning is in fact an important factor in determining behavior, but it is actually instantiated in a much simpler, nonsymbolic fashion than is usually assumed. Only by attempting to limit our preconceived ideas about the basis of behavior will we be free to explore a full range of possibilities.

To this end, the neural networks offer the most promising means for investigating robot control and planning because connectionist methodology allows the task demands rather than the designer's biases to be the primary force in shaping a system's development. At the initial design phase, connectionist methodology provides useful constraints by encouraging bottom-up construction. Input can come directly from the sensors and output can feed directly into the actuators creating a close coupling of perception and action. This interplay between sensing and acting, as opposed to a sense-REASON-act cycle, fosters a dynamic interaction between the controller and its environment that is crucial to producing reactive behavior which I believe is the essential precursor to planning.

Because adaptation is fundamental to the connectionist paradigm, the designer need not posit what form the internal knowledge will take or what specific function it will serve. Instead, based on the training task, the system constructs its own internal representations built directly from the sensor readings to achieve the desired control behavior. Thus the system's representations are automatically grounded in the environment.

Once the system has reached an adequate level of performance at the task, its method can be dissected and a high-level understanding of its control principles can be described. As Clark has noted, connectionism "inverts the official classical ordering, in which a high-level understanding comes first and closely guides the search for algorithms" (Clark, 1993). Through adaptation, the system's solution to the task emerges as it discovers the key features of the problem space rather than simply being the product of the designer's understanding of the domain. By actively restricting the extent of our presuppositions about planning, we may be able to develop truly autonomous robots with radically different forms of control and planning than previously conceived of[1].

1.3 An incremental approach to planning

Traditional models of planning, based on the deliberative worldview, have adopted a top-down perspective on the problem. They focus on the contemplative, internal qualities of planning at the expense of having a system that is grounded and connected to the world through its perceptions. My thesis takes the opposing, bottom-up perspective that being firmly situated in the world is the crucial starting point to understanding planning. The central hypothesis of this thesis is that the ability to plan developed from and was based on the more primitive capacity of reactive control. I will present a model that shows how the ability to plan could incrementally be adapted from an agent's immediate need to react.

In recent years there have been a number of different proposals for incrementally constructing intelligent agents (Brooks, 1986; Braitenberg, 1984; Ram and Santamaria, 1993; Schnepf, 1991; Waltz, 1991; Wilson, 1991). I will focus on Waltz's suggestions. Waltz notes that it will be some time before neuroscience can offer detailed wiring diagrams of the nervous system, and in the interim we need some plausible architecture for studying intelligence. He offers eight guiding principles to such an architecture. The first five of these principles lead up to planning, and the final three go beyond it. I have adopted the five principles relevant to planning and modified them to fit my own intuitions about the appropriate increments towards planning.

1. Use associative memory as the overall mechanism.
   $state \rightarrow action$

[1] I am not suggesting that the ultimate goal is a *tabula rosa* learner. Bias can never be completely eliminated, and it can often enhance learning. I am instead suggesting that bias should be explicitly noted so that its effect can be properly evaluated.

2. Populate the associative memory system with sequenced rote experiences.
3. Include mechanisms to automatically generalize across rote memories.
4. Include innate drive and evaluations systems to provide the robot with moment-to-moment guidance for its actions.
   $state + goal \rightarrow action + evaluation$
5. Include control structures to allow planning.
   $state + goal + plan \rightarrow action + evaluation$

These principles provide a foundation for a model of planning, but key issues remain unresolved. With respect to the first, second, and third principles, what sort of associative memory should be employed to ensure recognition of sequences and ease of generalization? With respect to the fourth principle, how should goals be specified to produce the appropriate type of motivation and what kind of evaluation should be given to best direct learning? Finally, and most importantly, with respect to the fifth principle, what will constitute a plan and how will plans develop?

The model to be presented here supplies one possible set of answers to these questions. The model is implemented as a connectionist network, which is fundamentally an associative engine. Chapter 2 addresses the first three principles with a discussion of connectionist architectures and mechanisms, describing how sensor states are associated with actions, how recurrent networks elegantly allow sequences to be learned, and how networks naturally perform generalization. Chapter 2 also describes other related proposals for constructing intelligent agents. Chapter 3 addresses the goal aspect of the fourth principle. The initial phase of the model is presented and several ways of providing goals are explored using a simulated robot in a restricted one-dimensional environment. Chapter 4 addresses the evaluation aspect of the fourth principle. Here, the model is extended to accommodate a real physical robot and two methods of providing evaluation are compared: a gradient descent algorithm and a genetic algorithm. Finally, Chapter 5 addresses the fifth principle related to planning. The robot's capabilities are extended to include sonar and the development and use of dynamic, context-dependent protoplans is examined. Chapter 6 concludes by tying the entire model together and discussing its significance.

2

Adaptation methods for learning control

2.1 Considerations for applying connectionism to control

Andrew Barto, in a review of connectionist approaches to the control of dynamical systems, argues that "because adaptive connectionist networks fit in the range between structureless lookup tables and highly constrained model-based parameter estimation, they seem well-suited for the acquisition and storage of control information. These methods suggest how new techniques for adaptive control can be developed which take full advantage of the possibilities for fabricating associative memory systems having high capacity, high speed, and the ability to usefully interpolate and extrapolate in real time" (Barto, 1989, p 31). Barto's review was based on applications such as controlling the production of a plant, but the problem is analogous to controlling a robot. In both problems the abstract goal of using resources in the most efficient way—to increase production in one case or to perform a set of tasks in the other—is not directly relevant to the moment-to-moment decisions that must be made. In addition, to react appropriately in a dynamic environment, a robot can obviously benefit from the type of memory he describes: one that is vast, quick, and from which generalizations are easily made.

The connectionist framework has a number of attractive properties, as suggested by Barto; however, there are some serious disadvantages as well. These strengths and weaknesses should be carefully considered before applying this framework to a particular problem. On the positive side, adaptive mechanisms are fundamental to the connectionist paradigm. The solutions learned by connectionist networks are an extremely accurate reflection of the statistical regularities that are present in the training environment. Not only do networks recognize and respond accurately to the trained examples, but as a side effect of the training, they can typically respond appropriately to novel instances which are similar to the trained examples. This ability to generalize from the training set is one of most attractive features of connectionist learning. Networks can also integrate top-down information (such as abstract goals) and bottom-up information (such as concrete sensory data) well. Finally, computation occurs in parallel, so it can be extremely fast.

On the negative side, in general, there is no guarantee that a network will find any solution, let alone an optimal one; and when there is convergence it can take a considerable amount of training, especially for large networks[1]. In addition, the type of generalization produced may not be what is needed for a given problem. The generalization exhibited is dependent on the training environment used. The more rich and realistic the training environment, the better the generalizations tend to be.

Because connectionism has only recently experienced a resurgence of popularity, there is currently a lack of strong theoretical principles to guide its application. Despite this, successful learning algorithms have sparked enthusiasm for this paradigm by producing good results in a wide variety of problem domains. The most common learning mechanism used in connectionism is a gradient descent procedure called back-propagation. Rumelhart, Hinton, and Williams, who popularized this algorithm, note that "although our learning results do not *guarantee* that we can find a solution for all solvable problems, our analyses and results show that as a practical matter, the error propagation scheme leads to solutions in virtually every case (Rumelhart et al., 1986, page 361, emphasis in the original).

Barto explains why he considers connectionism to be a valid paradigm for studying control even though its theoretical foundation still needs development:

> I have argued that the most distinctive character of connectionist learning systems lies not so much in their technical specifications as in the methodology with which they are applied... This experimental, heuristic approach is characterized by what I termed representational freedom and a willingness to plunge ahead when theoretical guarantees are lacking. It is easy to criticize this free-wheeling nature of much connectionist research, but theoretical guarantees are often obtained at the cost of extremely restrictive assumptions about tasks, which are almost always violated in practice.

[1] For some restricted circumstances—constant input/output mappings and periodic weight adjustment—convergence can be guaranteed.

> Certainly rigorous theory is important, and a valid criticism of any research is that it proceeds in ignorance of relevant theoretical frameworks and previous research, but an experimental methodology seems necessary for developing control applications involving complex nonlinear systems. (Barto, 1989, p 30)

Provably complete planners have been constructed that abide by classical planning's restrictive assumptions; David Chapman's TWEAK program is one example (Chapman, 1987). However, Chapman discovered that these restrictions made TWEAK almost useless as a real-world planner. TWEAK can generate complete plans because every possible action has a set of preconditions and postconditions associated with it. An action can only be applied when all of its preconditions are true; when an action is applied its postconditions are guaranteed to be true. But in this scheme, the effects of actions cannot depend on the situation in which they were applied, because then the postconditions could be different for every separate application. Thus relaxing these restrictions would again lead to the frame problem where effects of actions cannot be determined. With respect to robotics and planning, the theoretical gains obtained within the deliberative worldview have been made at the expense of realistic results. Adapting a grounded connectionist controller may prove to be a better foundation for the study of planning.

2.2 Connectionist architectures

The most common connectionist architecture is a three-layer feedforward network of processing units (see Figure 2.1). Each unit in any given layer (except the output layer) is linked by weighted connections to each unit in the layer above it. Various amounts of activation are applied to each of the units in the bottom layer, representing some particular pattern being presented as input to the network. This activation then flows across the connections to higher layers of the network, with the weights on the connections mediating the amount of activation that is passed on to successive units. The final pattern of activation present on the topmost layer is considered to be the output pattern produced by the network from the given input pattern.

A learning algorithm such as back-propagation can be repeatedly applied to the network, enabling it to learn to associate arbitrary pairs of input and output patterns by gradually adjusting the weights on the connections between units. The input pattern can be interpreted as representing perceptual information received from the environment and the output pattern can be interpreted as an action to be taken in response to that sensation. As a result of the training process, the network learns to

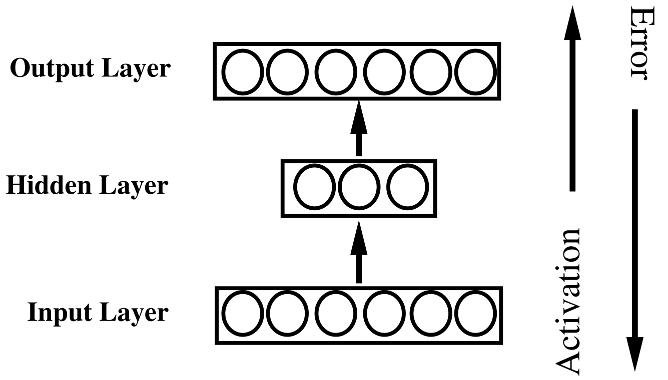

Figure 2.1: Standard Three-layer feedforward network architecture: This schematic diagram of a simple network depicts layers as rectangles, units as shaded circles, and weights as arrows. Instead of including all of the actual weights, bold arrows indicate that the the layers are fully connected (these conventions are followed throughout the thesis). For this particular network, there are actually 6×3 weighted connections from the input to the hidden layer and 3×6 from the hidden to the output layer, for a total of 36 weights. In addition, each unit in the hidden and output layers has an associated *bias*. The activation begins at the bottom in the Input Layer, proceeds upwards through the Hidden Layer, and ends in the Output Layer. Learning is accomplished by propagating errors from the Output Layer back through the network to the Input Layer.

recode each perception into different patterns of activation at the intermediate layer of units so that the appropriate action may be successfully generated at the output layer. The intermediate layer is termed *hidden* because it does not have direct access to the environment from either direction. This process of learning to recode input patterns into intermediate patterns of activation spread across the hidden layer amounts to the development of distributed internal representations of the input information by the network itself. The ability of connectionist networks to develop their own internal representations—or *hidden representations*—is an extremely important property of these mechanisms. See (Blank et al., 1992) for a thorough discussion of the unique aspects of connectionist representations.

One difficulty with the standard feedforward architecture is that when time is an important aspect of the problem to be modeled, it is not clear how to represent time in an efficient and useful way. Certainly for planning, timing information is crucial since plans involve temporal sequences of actions. A common method for accommodating time has been to represent time as space. In language processing, this entails giving an entire sentence to the input layer as a vector, where the first element in the vector occurred first in time and the final element occurred last in time. Elman contends that "a better approach would be to represent time implicitly rather than explicitly. That is, we represent time by the effect it has on processing and not as an additional dimension of the input" (Elman, 1990, p 180). To this end, a number of researchers have suggested recurrent architectures and new learning algorithms for them (Mozer, 1989; Pearlmutter, 1989; Williams and Zipser, 1989). Two simple recurrent architectures are due to Elman and Jordan and are shown in Figure 2.2 (Elman, 1990; Jordan, 1989). These are called recurrent architectures because processing loops are created by the backward connections.

In Jordan's version, the previous states of the output layer are made available to an additional bank of input units which he calls the state or plan layer. This gives the network a memory of its past output decisions. In Elman's version, the previous states of the hidden layer are made available to an additional bank of input units which he calls the context layer. This gives the network a memory of its own internal recodings of past inputs. The Elman-style temporal memory is more general than the Jordan-style because it stores past hidden unit activations which are not taught to assume specific values. The contents of a Jordan-style network's memory is restricted to the outputs of the specific training task. Both of these recurrent forms have offered elegant means of representing the temporal component inherent in many tasks.

The initial step towards a planning system will be an Elman-style network that maps the perceived sensor readings, the current goal, and the current context memory through the hidden layer to the next action as shown in Figure 2.3. A robot controlled by this network should be able to develop simple reactions in response to changes in

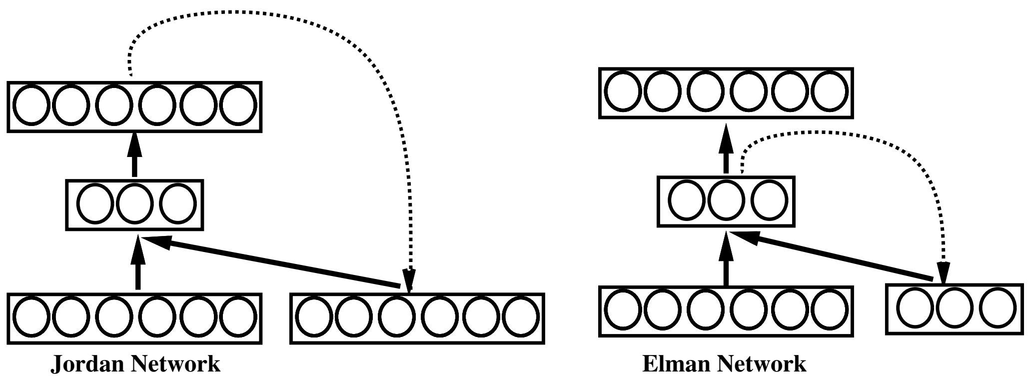

Figure 2.2: Simple recurrent network architectures: The bold arrow connections have adjustable weights. The dashed arrows indicate one-to-one connections between units. These one-to-one connections have a fixed weight of 1.0 and simply copy activations from the previous time step down for the next time step. The copy may be combined with a decayed version of the previous contents.

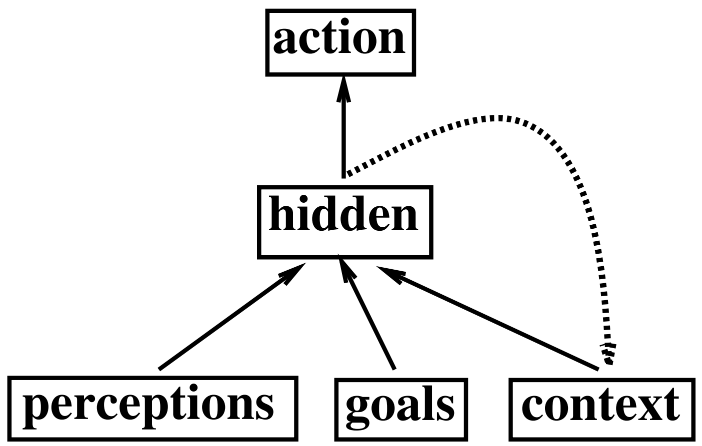

Figure 2.3: The initial architecture for the robot controller.

its sensor values. The next section describes how the weights for this control network will be learned.

2.3 Reinforcement learning

Standard back-propagation is an error correction method. Learning by error correction is one type of *supervised* learning procedure, because for each input pattern there must be an associated teacher or target pattern. Back-propagation compares the actual output produced by the network to the known target pattern to determine the error that is then used to adjust the weights. However, when the task to be learned does not have specific targets, but only more abstract measures of goodness (as is the case for robotics problems), a standard error correction method cannot be used.

Consider again the simple robot described in the first chapter. Its task was to reach a light while avoiding the boundaries of the environment. Suppose that during its initial learning it bumps into one of the walls. Any action that moves it away from the wall should be rewarded, while any action that persists in causing it to contact the wall should be punished. There is not necessarily one right or wrong action for a given situation, and even if there were, it is probably not known *a priori*. For these types of domains, reinforcement learning, another supervised method of learning, works well. During reinforcement learning the kind of supervision given is much less specific than for error-correction learning. A reinforcement procedure determines whether the output should receive positive, negative, or no reinforcement. Essentially this is equivalent to teaching by telling the learner whether its output was good, bad, or indifferent. The magnitude of the reinforcement value can also reflect its degree of goodness.

In this thesis, two very different kinds of reinforcement procedures have been used to adapt the weights of the control networks—one is a local method while the other is a global method. The local method is a special back-propagation algorithm that updates the weights immediately upon receiving reinforcement from the environment. The global method is a genetic algorithm which tests out a given network in the environment for an extended period of time. From this test it obtains a global fitness measure that is then used to bias the subsequent adaptation. Each of these methods is described in the following sections.

2.3.1 Local method: Complementary Reinforcement Back-Propagation

The local method is a modified version of the complementary reinforcement back-propagation (CRBP) learning algorithm (Ackley and Littman, 1990). Back-propagation learning requires precise error measures for each output produced by a network so that gradient descent on the error can be performed. CRBP provides these exact error measures from the abstract reward and punishment signals as described below.

A forward propagation of the input values produces a real-valued search vector S on the output layer. Each of these activations is interpreted as the probability that an associated random bit takes on the value 1. From these probabilities a binary output vector O is stochastically produced. If O is rewarded, then learning should push the network towards this vector, so the error measure $(O - S)$ is back-propagated. If O is punished, then learning should push the network away from this vector, but the appropriate direction is not clear. Punishment only provides negative evidence, it does not offer any other guidance. CRBP chooses to push the network directly toward the complement of O, using the error measure $((1 - O) - S)$. In this way, in similar situations rewarded outputs will be more likely to occur again and punished outputs will tend to produce the opposite effect.

Another feature of CRBP is that there are different learning rates for reward and punishment. When the network is rewarded, we can be confident that we have good information to learn from because the current state led to positive reinforcement. Therefore, we should use a high learning rate. In contrast, when the network is punished, we know that the current state is not desirable, but we can only arbitrarily pick the complement state as our target. There is no guarantee that the complement is a good choice, so our learning rate should be lower.

The original version of CRBP was applied to static problems and was designed for feedforward networks. I modified CRBP to work with dynamic problems and simple recurrent networks. To my knowledge, this research is the first application of CRBP to time-dependent domains and networks. In the original static version of CRBP, a reward learning rate ten times larger than the punishment learning rate works well. In testing the viability of applying CRBP to dynamic problems, this high ratio of the learning rates proved to be problematic. In dynamic robot domains, the reward to punishment ratio tends to be higher than for static domains, making such a large disparity in the learning rates infeasible. Therefore, in the experiments reported in this thesis (unless otherwise noted), the reward learning rate was only three times larger than the punishment learning rate.

Implementation details

Training begins by initializing all the weights in the network to small random values. The following set of steps is considered one *cycle* of processing and is iterated until some limit is reached. The limit ranged from $100,000$ to $500,000$ cycles depending on the particular experiment.

1. Get input from the sensors.
2. Forward propagate input to produce real-valued vector S.
3. Stochastically determine binary-valued vector O from vector S.
4. Execute action vector O on robot.
5. Determine reinforcement.
6. Determine error vector E.
   If rewarded then $E = O - S$
   If punished then $E = ((1 - O) - S)$
   If no feedback then $E = 0$
7. If $E \neq 0$ then back-propagate E.
   Use higher learning rate for reward than punishment.

2.3.2 Global method: Genetic Algorithm

Although genetic algorithms (GAs) have been applied to machine learning problems, they are not generally seen as reinforcement procedures. Instead they are primarily viewed as search-based function optimizers. Goldberg provides a good introduction to both types of applications (Goldberg, 1989).

Genetic algorithms are based on the theory of natural selection in evolution. GAs work on a *population* of individuals, where each individual represents a possible solution to the given problem. After the initial population is randomly generated, the algorithm evolves the population by creating new individuals through the recombination of information from parents in the current population; this is called *reproduction*. The key to the success of this process is in the *selection* of parents from the current population. Individuals with above average fitness are more likely to be chosen for reproduction than individuals with below average fitness. *Fitness* is some measure of the goodness of an individual that the GA is trying to maximize. Through selection, GAs allocate more trials to regions in the problem space that contain above average solutions. According to Whitely, "genetic algorithms are capable of performing a global search of a space because they can rely on hyperplane sampling to guide the search instead of searching along the gradient of a function" as back-propagation does (Whitley et al., 1993).

Combining genetic algorithms and connectionist networks

There are many interesting possibilities for applying genetic algorithms to connectionist networks. GAs have been used to find good initial network weights, to tune network learning parameters, to determine network structure, to evolve network learning algorithms, and to learn network weights (Belew et al., 1992; Harp et al., 1989; Harvey, 1993b; Chalmers, 1990; Whitley et al., 1993). It is the last option that will be used here: the network architecture is fixed and the GA works to adapt an appropriate set of weights.

Applying genetic algorithms to networks has not been as straightforward as other types of GA applications. Traditionally individuals in GA populations have been represented as bit strings, but weights in networks are typically real-valued. By working at the level of bits, the GA has no knowledge about the semantics of a problem. This is considered an advantage because the GA's success cannot be linked to any built-in knowledge about a particular domain (Belew et al., 1992). Although there are methods for translating real values into bit strings appropriate for GA processing, for the robot domains studied here, individuals are represented as real-coded vectors of weights. Real numbers are certainly more abstract than bits, but there is still very little semantics about the robot revealed in this real-coded representation. Thus this real encoding still limits the effect that designer bias can have on the outcome of the learning.

Another tradition in GAs is that the primary form of recombination is *crossover* and the secondary form is *mutation*. In crossover, parents contribute complementary portions of their bit strings to an offspring. In mutation, with some small probability, randomly chosen bits in the offspring are flipped.

When evolving networks, crossover would create a new set of weights by taking some weights from one successful network and the rest from another successful network. However, these two networks could be using very different strategies for solving the problem. Creating an offspring by recombining portions of their weights may result in an extremely poor alternative solution. In fact even if the two networks are employing the same strategy, it is probably instantiated in the weights in very different ways. So again, recombination may be quite unsuccessful. Networks tend to solve problems in a distributed, holistic fashion and thus may not have useful building blocks to contribute to a recombined solution. Due to the problematic nature of crossover for network representations, this form of recombination is not used here (although crossover has been used in other GA applications to neural networks—for a good discussion of the potential problems and solutions see (Belew et al., 1992; Whitley et al., 1993)).

Instead, new individuals are created solely by mutation. Harvey notes that there has been "surprising success (in some circumstances) of what has come to be called *naive evolution*; i.e. mutation only, contrary to normal GA folklore which emphasizes the significance of crossover" (Harvey, 1993a). Further, he found that the optimal mutation rates were between one and two mutations per individual and this was nearly invariant over the length of the individual's representation.

Implementation details

Each individual in the GA's population is a possible control network for a robot. Therefore the fitness function should measure the effectiveness of a network for controlling the robot in its particular task. The fitness is determined as follows. A random situation is selected for the robot. The particular network then controls the robot for 50 actions starting from that point. Each action receives either a reward or punishment and the sum of the feedback over all 50 actions is calculated. Another random situation is chosen and the same process is done again. The fitness is the average feedback received over these two random starts. Allocating only 100 actions to evaluate each network results in a fairly noisy measure. However it is better to obtain quick, rough estimates of fitness and to allow the GA to consider many candidate solutions than it is to attempt to obtain highly accurate evaluations with a smaller population size (Fitzpatrick and Grefenstette, 1988). GAs have proved to be quite adept at finding good solutions in very noisy environments.

Processing in GAs is measured in terms of *generations*. A generation is completed when a new population has been created through reproduction within the old population. The number of reproductions needed to replace every individual in the old population equals the size of the population. To choose parents for the next generation, a technique called *tournament selection* was used (described below). Iterating the following set of steps 250 times (the standard population size) constitutes a generation. Each GA run typically consisted of 1000 generations.

1. Randomly choose two individuals from the population to compete in a tournament.

2. Declare the individual with the higher fitness as the winner.

3. Replace the loser by a mutation of the winner.
   Create mutation by randomly picking two weights in the winner and updating them by random values between -5 and $+5$.

4. Determine fitness of the new individual.

2.3.3 Comparison of methods

GAs tend to be a very robust method because they operate on a population of possible solutions rather than just a single solution as is done in CRBP. Using CRBP, there is much more potential for getting stuck in local minima. For example, if CRBP begins with a poor set of initial weights it may never be able to converge on a solution. Whereas a GA, given enough processing time, can more reliably find at least a reasonable solution.

In terms of robot actions executed, however, this processing price can be very steep. For the GA, the total number of actions performed in a single run is:
$actions = popsize \times generations \times starts \times steps = 2.5 \times 10^7$
where the population size is 250, the maximum number of generations is 1000, the number of random starts is 2, and the number of steps per start is 50. In contrast, the CRBP runs performed at most 5×10^5 actions (50 times fewer actions).

Perhaps the most significant difference between these two methods is in the immediacy of the reinforcement used. CRBP was designed to learn from direct feedback—an action is executed and an evaluation of its goodness is immediately required. GAs learn from indirect feedback—a sequence of actions is executed and some overall measure of fitness is returned upon completion. Thus GAs are an inherently delayed reinforcement procedure. Experiments in Chapter 4 will examine this difference in the adaptation methods and test whether CRBP is able to learn from delayed reinforcement.

2.4 Related work

The first chapter illustrated the difficulties with accommodating real perception and action within the deliberative worldview. We have just defined an initial neural network model for tackling the problem of robotics control from the alternative view of adaptation from limited designer bias. There have been several other neural network models proposed in response to problems with the deliberative worldview that can also be seen as lying within this alternative view.

2.4.1 Adapting pure reactivity

One approach has been to eschew memory-based control entirely. Pfeifer and Verschure have discussed several reasons why reactivity should be the sole basis for autonomous agent design (Pfeifer and Verschure, 1992b). In their view, memory use is linked to goal directedness and plan following, which they suggest may be unnecessary for producing structured behavior. Traditionally, goal-directed systems resort to plans, which in turn have required symbolic world models. These world models are based on static designer-defined ontologies that must be determined in advance and are difficult to adapt. To use systems of this sort, task descriptions must first be translated into symbolic goal structures. Yet there is no principled way to perform this translation and the end result is ad hoc systems.

They propose a model based on distributed adaptive control (DAC) that can perform tasks without resorting to goals, plans, or internal world models. The DAC model, shown in Figure 2.4, is a neural network based on classical conditioning principles that learns to integrate pre-wired reflexes with sophisticated sensors (Verschure et al., 1992). This network has no recurrent connections and is essentially feedforward except for the lateral connection between the collision and target detection groups.

Prior to learning, the system's pre-wired reflexes encode how to respond to collisions and how to approach a target while avoiding obstacles in the near vicinity. For instance, if contact is detected to the left, the model will automatically trigger a `reverse-right-turn` action. These basic reflexes are just one part of what they term a *value scheme* that is seen as a representation of the type of information passed on through evolution which constrains and directs the process of development. This value scheme includes information about appropriate parameter settings, the properties of the sensors, and the operation of the actuators.

Robots controlled by the DAC model have learned to navigate amongst obstacles to locate a target. Initially these robots depend on the pre-wired reflexes, but as associations are made between the range finder and the primitive sensors, the robot develops more sophisticated strategies such as wall following.

The designer's role in this framework is clearly demarcated: to define the value scheme to enable the agent to establish an interaction with the environment so that the desired behavior can emerge. Starting from an initial set of biases the system is free to develop its own associations. However, the designer plays a larger role in the

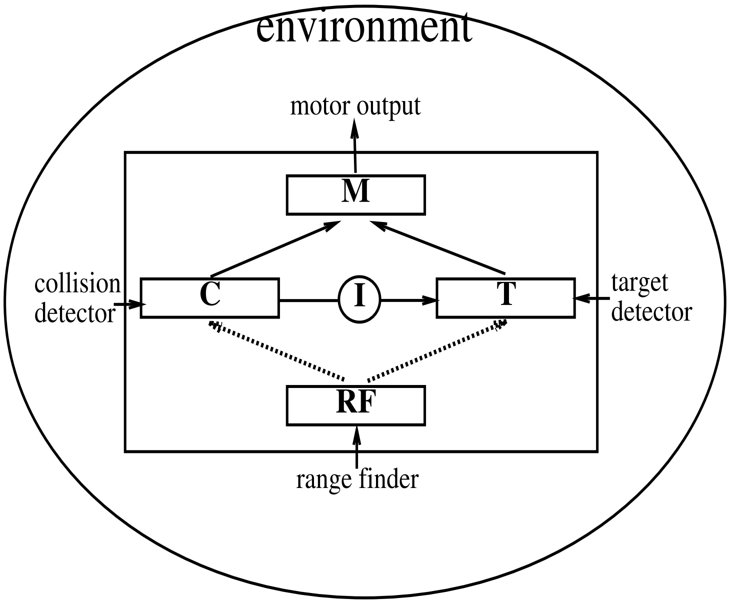

Figure 2.4: Schematic drawing of the distributed adaptive control architecture. Small rectangles depict groups of neurons. The solid lines represent prewired connections, while the dashed lines represent modifiable connections via Hebbian learning. There is an inhibitory unit (shown as **I**) between the collision detection and target detection groups to create a built-in preference for avoidance over approach.

DAC model than in the model to be presented here. In my model, the only initial associations are determined by the random starting weights[2]. My system must essentially begin learning from scratch. Although most organisms do begin development with many built-in constraints, I wanted to explore how even very primitive associations might form. Consider for example the DAC model's target detection. The robot is equipped with two sensors placed at -90 and +90 degrees from the robot's center that can calculate their distance to a target. The difference between these two sensor readings is used to determine whether a target is to the left or the right. In some organisms, this level of orientation ability might not be built-in, but might have to be learned.

A second and more significant difference of the DAC model is that it can only act on its immediate sensory states while my model maintains a short-term memory of the recent past. The DAC model was designed to be purely reactive with no potential for planned action. In contrast the ultimate aim of my model is to understand how planning ability might have evolved so the building blocks of plan-based action, such as short-term memory, are provided.

I concur with Pfeifer's and Verschure's criticisms of traditional deliberative models. However, I take exception with their overarching assumption that goals and plans must be represented symbolically. This thesis proposes instantiations of both goals and plans that are essentially non-symbolic. It further suggests how planning could be grounded in the environment without recourse to symbolic world models. The question of whether goals and plans play a role in determining behavior should be independent from one's stance in relation to the deliberative worldview.

2.4.2 Evolving dynamic state

Rather than eliminating internal state as an input to the system, a second response has been to design network architectures that will develop their own dynamic state. The Elman and Jordan networks already described are examples of this approach. Below, Beer and Gallagher discuss why it is advantageous for controllers to maintain state.

> A significant problem with some of the other control mechanisms currently being explored is that they are purely functional in nature. The response

[2]In both the DAC model and my model, bias is also reflected in the chosen architecture. Both models strive to create a close coupling of perception and action, while my model further emphasizes contextual memory.

> of the agent at each instant is determined solely by the environmental stimuli it encounters *at that instant*. This is undesirable because such agents are constantly pushed around by their environment. They cannot take any initiative in their interactions and therefore exhibit no true autonomy. This problem is intrinsic to some of the architectures (e.g. feedforward neural networks), but is merely a frequently taken option with the others. In contrast, because dynamical neural networks maintain state, their response to identical environmental stimuli can differ at different times. Their activity exhibits a certain "inertia" independent of their immediate environmental context. (Beer and Gallagher, 1992, p 115)

Beer's dynamical neural network model shown in Figure 2.5 is quite similar to the Elman-style architecture being used as the basis for this research. Beer's model contains only two layers where the second layer is fully recurrent (i.e. every unit is connected to every other unit, including itself). An Elman-style network also contains a fully recurrent second layer, although it is not always depicted in this fashion. In Beer's model there is an additional parameter, called a time constant, that is associated with each second layer unit to create the dynamic state. When this time constant is high, the previous activation of a unit will strongly influence its current activation. Symmetrically, when this time constant is low, the current activation will depend almost solely on the immediate incoming activations. The primary difference between these two models is that in an Elman-style network all the units are updated synchronously, while the time constants of Beer's model allow each unit to integrate information at its own continuous rate. This fluid approach to time may enable Beer's model to become more finely tuned to the intricate dynamics of an environment. However, his experiments to date have explored environments with fairly simple dynamics, so the advantage of time constants over an Elman-style architecture has not yet been demonstrated.

Another difference between this work and my own, is that Beer has only used genetic algorithms to adapt his networks, while I have also explored gradient descent learning. Starting with a fixed architecture, Beer and his students used genetic algorithms to evolve the weights, biases, and time constants for dynamical network controllers that perform a wide range of behaviors including chemotaxis (orientation along a gradient), insect-like locomotion, landmark recognition, reactive navigation, predator avoidance, sequence learning, and sequence generation (Beer and Gallagher, 1992; Beer, 1990; Yamauchi, 1993; Yamauchi and Beer, 1994a; Yamauchi and Beer, 1994b). Much of this work was done concurrently with my own and is quite similar in flavor. The primary difference is in my emphasis on planning.

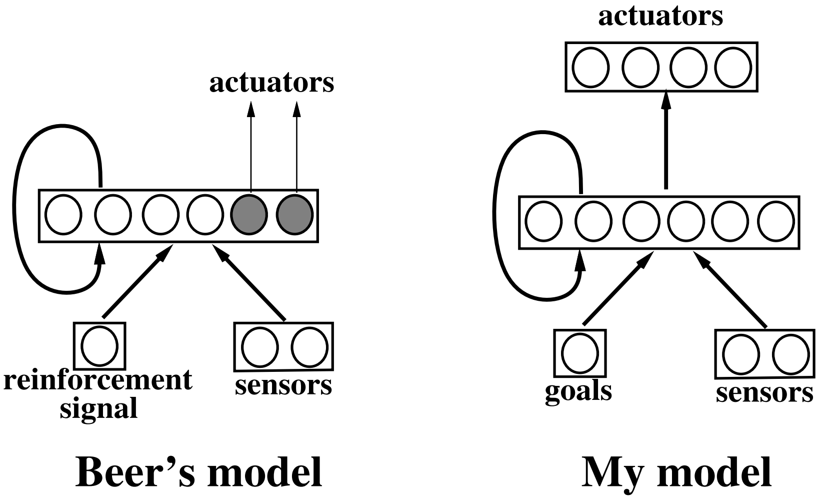

Figure 2.5: Comparing Beer's and my network architecture as applied to robotics control. Both versions contain a fully recurrent second layer of units. Beer's version does not contain an output layer per se. Instead certain units are designated for output (shaded here). In some of Beer's work, the second layer also receives input from a binary reinforcement signal.

2.4.3 Evolving dynamic structure

In all of the network models presented to this point, the architecture has remained fixed while the parameters, such as weights, have been modified to improve performance. Yet by fixing on a particular architecture we have bounded the set of possible solutions. Harvey, Husbands, and Cliff take the stance of limiting designer bias one step further by suggesting that the architecture itself should be allowed to evolve over time (Harvey, 1993b). They argue that determining an appropriate network architecture for a particular task is not always easy. For instance, one must determine what hidden layer size will be large enough to allow the task to be learned but small enough to produce successful generalization. In addition, there may be tasks where we don't even have enough information to use reinforcement learning. Thus, they conclude that the automatic development of noise-tolerant, dynamical, neural network structures through evolution is the most promising route towards creating control systems.

Their networks have a fixed number of input units, one for each sensor, and a fixed number of output units, two for each motor, but the number of hidden units and the pattern of connections is not prespecified; the topology is completely unrestricted. There are two types of connections between units: inhibitory and excitatory. An inhibitory link is considered to be infinitely negative and so has veto power over the units it is connected to (Cliff et al., 1993). Infinitely negative connections may seem to be too strong a constraint, but the designers of this model claim that there is a similar phenomena found in invertebrate nervous systems.

Their ultimate goal is to develop visually guided robots and they have begun to explore this using ray tracing techniques, but much of their preliminary investigations used robots and environments similar to those examined here. One robot was circular and was equipped with front and back bumpers for sensing collisions and several whiskers for anticipating collisions. The environment was a circular enclosure where the only obstacles were the boundaries. In the genetic algorithm the robot's fitness was a function of its proximity to the center of the enclosure. The evaluation function was chosen to see how well their evolved networks could keep the robot away from the walls. The resulting control networks produced circular paths which frequently entered the high fitness area at the center of the environment. However, determining how this behavior was produced was not a straightforward matter.

One disadvantage of allowing network controllers to evolve unconstrained structures is that the resulting architecture may be extremely difficult to analyze. To elucidate the essential characteristics of the controllers, they developed a technique for reducing the networks to a simpler though functionally equivalent form. First units with no outgoing links were removed. Then time series plots of sensor, neuron, and motor activities were used to further eliminate units that were deemed to play no role in the behavior. In this way a network with 13 units and 34 connections was reduced to 8 units and 21 connections (Husbands et al., 1993). Despite their initial success, as the size and complexity of these amorphous structures grows it will become increasingly difficult to analyze their mechanisms. In response to this, they claim that "with an evolutionary approach it may not be necessary to analyze how it works, but rather one should assess how good is the behavior it elicits" (Harvey, 1993b).

Ultimately I agree with Harvey, Husbands, and Cliff that the neural structure itself must be evolved. Yet, I believe that there is still much to be learned about potential control mechanisms from simple recurrent network architectures that might be obscured within unconstrained architectures.

3

Experimenting with goals

> A general intelligent system must somehow embody aspects of what is to be attained prior to attainment of it, i.e. it must have goals. Symbols that designate the situations to be attained (including that it is to be attained, under what conditions, etc.) appear to be the only candidate for doing this. (Newell, 1980)

A redefinition of plans as dynamic, context-dependent entities should in turn lead to a redefinition of goals, since the two are so inextricably linked. This chapter explores the use of goals with connectionist controllers and will provide some insights into the further development of the planning model.

As Newell's quote above suggests, traditionally in AI, goals have been represented as complete state descriptions that define the desired results in detail. In the blocks world, the goal to create a stack of blocks A, B and C, might be represented as follows:

```
(and (on A B) (on B C) (on C table) (clear-top A))
```

Research in reactive planning has demonstrated that symbolically represented goals can be more flexible than pure state descriptions. For example, consider the type of goal Friby employed in his reactive action packages (Firby, 1989). His system controlled a delivery truck that could be attacked by roving enemies. To determine whether the truck was prepared for battle the following goal was used:

```
(and   (location ?weapon weapon-bay)
   (class ?weapon weapon true)
   (quantity-held ?weapon ?amount)
   (> ?amount 0))
```

This type of goal contains variables (indicated by the question marks) and can be repeatedly applied to varying circumstances in the world.

Both symbolic goal forms—the traditional state description and Firby's more flexible reactive style—encode much of the relevant information needed to tackle the given problem. In the blocks world case, the desired location of each block is noted. In the delivery truck case, to be ready for battle means that the truck needs to have a weapon and ammunition on board. Both representations provide possible sub-goals towards achieving the primary goal and thus reflect designer bias. What sort of goal could be employed to limit designer bias more?

I propose the opposite tack: Rather than attempting to concretely describe the goal as a state, provide the system with only an abstract cue and allow it discover the true goal through learning. This form of goal is essentially a *trigger*—it can be used to initiate a process or reaction. Thus instead of imposing highly constrained external goals, this proposal supports my focus on adaptation from limited designer bias by giving the system the power to determine its own internal goals.

It may be the case that a connectionist controller will not be able to benefit from such abstract goals. To investigate the effect of triggering goals on behavior, a number of experiments were performed where the availability of an abstract goal in the input was varied. The results reveal that the addition of abstract goals profoundly affects every aspect of the control networks—the final behavior, the pattern of the learned weights, the use of recurrent memory, and the course of learning. Furthermore in certain cases, abstract goals do enhance a connectionist controller's ability to perform a task.

3.1 Starting small: A one-dimensional world

These initial experiments were performed with the intent to gain insight not only into goal use, but also into the inner workings of connectionist robot controllers. To this end, a minimalist robot, environment, and network architecture were constructed[1].

The robot and environment are depicted in Figure 3.1. The environment is one-dimensional and ten units in length with a light centered at the origin. The robot can perform three actions: `left`, `right`, or `not-moving` and is equipped with three sensors: two to detect collisions with the left and right boundaries and a single light sensor. Each `left` or `right` action moves the robot $0.7+random(0.3)$ units in distance (in the positive direction for `right` and in the negative direction for `left`). The light

[1]The use of the word *robot* may be a bit misleading at this point—the experiments described in this chapter were performed solely on a simulator. However, the experiments discussed in Chapter 4 were done on an actual robot.

falls off linearly as the distance from the origin increases:

$light = 1 - (abs(position)/5)$

where *abs* is the absolute value. For instance, the light readings are 1.0 at position 0, 0.8 at position +1 or -1, and 0.0 at position +5 or -5.

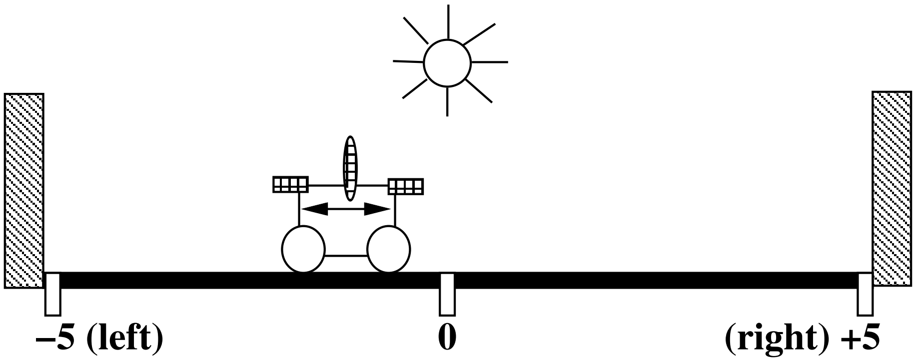

Figure 3.1: Simple one-dimensional robot and environment. The robot moves left to reach the negative side and right to reach the positive side.

Note that the robot is not given any information about its current position. Since the light readings are identical for the corresponding positions on each side of the origin, the only way for the controller to determine on which side the robot has been placed is to touch a boundary or to correlate the robot's movements with changes in the light gradient.

Although the robot and environment are quite simple, a task was defined with both a reactive level and a goal-based level. At the reactive level, the robot was punished any time it contacted one of the boundaries or was not moving. This is rudimentary navigation: the robot must learn to continually explore the environment while avoiding the boundaries. There are no goals provided for the reactive aspect of the task.

At the goal level, the robot must either seek or avoid the light depending on the current goal. A positive value for the goal indicated that the robot should seek out the light until the $light > 0.9$ which means it must be within one-half of a unit from the origin. Once this is accomplished, the goal automatically switched to a negative value, indicating that the robot should avoid the light until the $light < 0.2$ which means it must be within one unit from a boundary. Successful avoidance switched the goal back to seek-mode again. The goal varied in this periodic manner throughout the task, seeking was always followed by avoiding and so on. For the goal-based portion of the task, the robot was punished any time it failed to follow the light gradient appropriately for its current goal. For example, if the goal was to seek the light and the robot moved away from the light, it was punished, but if it moved towards the light it was rewarded.

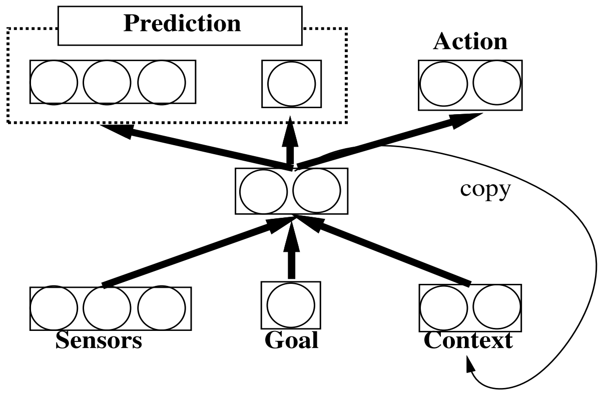

Figure 3.2: Control network used for one-dimensional environment.

The robot is controlled by the simple recurrent network shown in Figure 3.2. This network gathers input data from the robot's sensors and combines this information with the abstract goal and recurrent memory to determine an appropriate next action. It has the additional task of trying to predict the sensor readings and goal that will result from its chosen action. This prediction training facilitates the network's use of the recurrent memory. The size of this memory was deliberately limited to two units so that its contents could be easily traced over time in two-dimensional graphs.

A sophisticated solution to this one-dimensional task would require the controller to attend to the changing goals, contacts with the boundaries, and the light gradient. The first two features (goals and walls) are directly available on the input, but the light gradient depends on memory of the previous light reading. With only two hidden units, it is not possible to monitor all of these parameters. As will be seen below, the adapted controllers focus on reacting to the goals and walls and ignore the light gradient in their solutions.

At the input layer, the first set of units holds the sensor values. The touch sensors are digital, responding with a one if the robot is in contact with a boundary and a zero otherwise. The light sensor is analog, as already described, varying from 0.0 to 1.0 depending on the robot's position relative to the light. The next set of one unit encodes the abstract goals of the robot: +1 for seeking the light and -1 for avoiding the light. The last set of units, the context, provides a short-term memory of the network's past states. Activations from the hidden layer on the previous time step are copied into the context units directly. At the output layer, there is a set of two action units: the first designates a `left` action and the second a `nright` action. If neither unit or both units are on then the robot does not move.

3.2 Varying the presence of abstract goals

Three goal variations were examined:

1. *Constant*—the goal is constantly present as input.
2. *One-step*—the goal is present as input for the single time step after the previous goal has been accomplished to indicate the new goal.
3. *Implicit*—the goal is never present; the goal unit remains in the network but its activation is always zero.

To determine the baseline behavior of the network architecture under the three variations, a set of non-learning experiments were performed. For each variation, ten networks with different initial random weights were used to control the robot for 1,000 actions. The average percent of the actions punished, across the ten trials, was used as the performance measure for comparison. The results are shown in Table 3.1. Note that prior to learning all the network controllers are being punished a majority of the time.

Variation	Avg %pun	SD
Constant	78.8	4.0
One-step	78.0	4.1
Implicit	83.7	5.3

Table 3.1: Baselines: Summary of performance results for goal variations prior to learning

For each variation in the learning experiments, twenty networks with different initial random weights were trained for 30,000 actions using the CRBP algorithm previously described. Again the average number of actions punished was used to compare their performance.

Variation	Avg %pun	Best %pun	Worst %pun	SD
Constant	37.2	6.6	96.5	27.6
One-step	11.3	1.7	64.7	11.3
Implicit	15.6	11.6	26.2	3.4

Table 3.2: Summary of performance results for goal variations after learning

Table 3.2 summarizes the learning performance results. Learning has significantly improved performance, relative to the baselines, in all three variations. Perhaps surprisingly though, having a constant goal is not always beneficial—on average the network performs better with no goal at all than with a constantly present goal. The most successful way to provide a goal seems to be intermittently as the goal changes. Despite high variance in the performance levels, these goal variations produced statistically significant results in a three-way comparison using analysis of variance testing ($p < 0.01$). The cause of this high variance will be discussed below as the results for each goal type are described in detail.

3.2.1 Constant goals

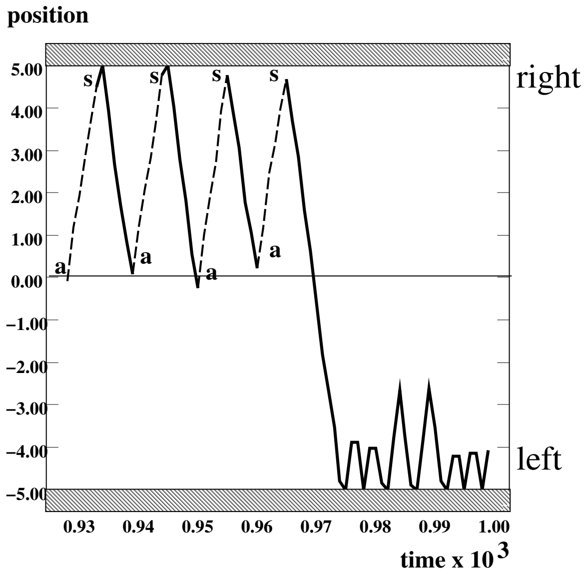

Figure 3.3: Constant goal: behavior. The one-dimensional world is shown in two-dimensions with time. The positive end of the environment is reached by going right and the negative end by going left. Avoid paths are dashed lines and seek paths are solid lines. Each letter indicates the start of a new goal—**a** for avoid and **s** for seek. Recall that the light is located at the origin.

When a constant goal is present, the most prevalent strategy learned by the networks was to remain on one side of the environment and associate one action direction with avoiding and the other with seeking. For example in Figure 3.3, to avoid the light the constant goal controller moves the robot to the right and to seek the light to the left. Sometimes the constant goal controller does not immediately switch directions when the goal switches, as in the first two seek instances. Instead, contact with the boundary triggers the change in direction in these cases.

This strategy is quite successful most of the time. A problem arises however, when the robot oversteps the light during a seek goal as can be seen at the end of Figure 3.3. Recall that the robot cannot control its speed. It always moves at least 0.7 units plus some random noise of at most 0.3 units. Suppose the robot is at position +0.6 which creates a light reading of 0.88 (under the limit of the seek goal of 0.9). So the constant goal controller continues to move the robot to the left to increase the light reading. If this step is close to the maximum size of 1.0 units then the robot will be on the other side of the light at position -0.6, but still only have a light reading of 0.88. So it continues to move left. The light reading will now fall, but since these circumstances occur relatively infrequently, the constant goal controller does not learn to adapt to them and continues to move left entering into an infinite loop next to the left boundary.

These situations of overstepping the light during the seek goal are the primary source of the high variance in the performance metric. Controllers of any goal type that happen to overstep the goal during the testing will receive much more frequent punishment than other controllers using the same basic strategy. The constant goal networks are the most susceptible to falling into ineffective behavior of this sort and an examination of the weights that produced the behavior shown above will reveal why this is so.

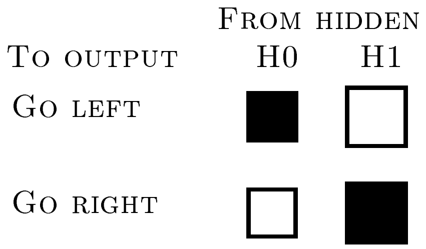

Figure 3.4: Constant goal: hidden to output weights. To simplify the presentation, the weights associated with prediction have been excluded in this and later figures.

To understand a control network's weights it is often best to begin at the output layer and discover how the hidden units are associated with the robot's actions. Figure 3.4 depicts the weights from the hidden units to the action units, where the the size of each square reflects the magnitude of the weight and the color reflects the sign (black for inhibitory, white for excitatory). We can see that when $H0$ is active it causes the robot to move right (which is the avoid strategy for this network), and H1 has the opposite effect. Now we can go on to examine how the inputs connect up with these two hidden units in Figure 3.5.

$H0$ creates right motion and as we would logically expect, the left sensor activates

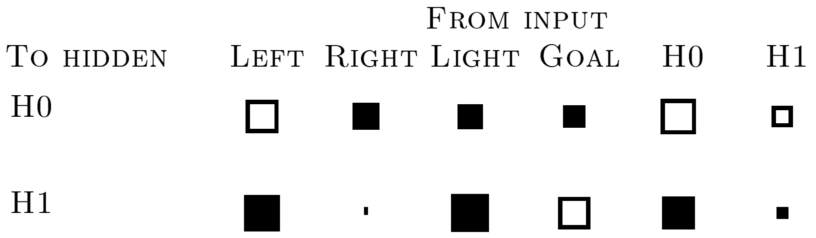

Figure 3.5: Constant goal: input to hidden weights. The context layer weights are shown here as weights from $H0$ and $H1$ to $H0$ and $H1$.

$H0$ and the right sensor deactivates $H0$. Furthermore, because this network has learned to respond to the avoid goal by going right, a positive goal value deactivates $H0$ while a negative value activates it. The pattern of $H1$'s weights are approximately complementary to $H2$'s (except in two cases: for the right sensor and the light). Looking only at the weights, it appears that the light serves an inhibitory function for both hidden units. However, each hidden unit also has a positive bias (not shown) that cancels out the negative effect of these weights. Essentially then, the light plays no role in this constant goal controller's action decisions.

How does the constant goal controller end up in an infinite loop when it oversteps the light? The positive goal value activates $H1$ causing the robot to move left. If the seek goal isn't satisfied the robot will continue to move left and contact the left boundary triggering the left sensor. This in turn will activate $H0$ causing the robot to move right. Notice that $H0$ sends positive feedback to itself through the context layer so it can sometimes sustain its activation for a few steps. But since the seek goal is constantly present on the input, it is still exciting $H1$. Inevitably $H1$ will become active again and then send inhibitory feedback through the context layer to $H0$. Sometimes both hidden units will be on for a few steps causing the robot to remain stationary (seen as a flat section in its path in Figure 3.3). Eventually $H1$ will become solely active causing the robot to move left into the boundary thus restarting the cycle again.

We've seen how the network has implemented a control strategy in its weights; let's now consider how the context memory is being used. Figure 3.6 shows the activations of the two hidden units during the successful portion of the behavior shown in Figure 3.3 above. As was predicted from examining the weights, $H0$'s activation is strongly correlated with the avoid phase of the task and $H1$'s activation is correlated (though less strongly) with the seek phase. The two seek paths in the hidden space that contain a loop were created during the two physical paths in the

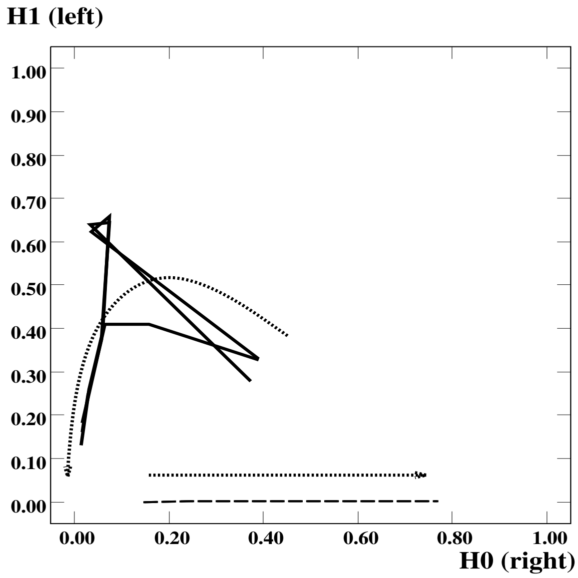

Figure 3.6: Constant goal: hidden activations through time. Avoid paths are dashed lines and seek paths are solid bold lines. The dotted lines with arrows were added to show the direction of movement through the hidden space over time.

environment where the right boundary was contacted before a switch in direction was made.

In Figure 3.7 we can see what happens to the context memory during the inappropriate cyclic behavior that occurred after the robot overstepped the light. Initially the hidden activations mark out a similar path to the previously shown seek phases, moving toward the origin, but then they begin cycling in the opposite direction never returning to the learned dynamics.

One final means of dissecting the network's processing is to examine the weight changes over the course of the training as shown in Figure 3.8. In the first 5,000 steps of training, the error drops fairly steadily as the weights from the hidden layer to the output layer experience the largest change. By looking at the behavior produced during this portion of the training we can deduce what the network is learning. Before the network has even managed to keep the robot consistently moving or to move the robot away from the boundaries after contact, it has strongly associated each goal with a direction of movement. This occurs within the first 1,000 steps of training. By the 5,000 step mark, the network has learned the reactive aspects of the task: the robot is rarely stationary and typically responds to boundary contacts immediately. Most of the subsequent training is ineffective—an upswing in weight changes (dominated by the context layer) is closely followed by a rise in error and a then drop in weight

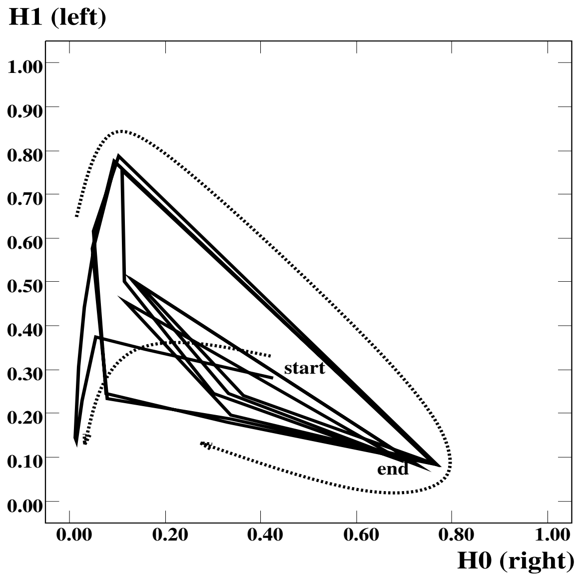

Figure 3.7: Constant goal: cyclical hidden activations during seek behavior when light was overstepped

changes.

The root of the problem seems to be that when a constant goal is available to the network during adaptation, the constant goal controller becomes dependent on this goal quite early in the training and then either ignores or is unable to attend to other available environmental cues for guiding behavior. For instance, in the constant goal network just examined, the light readings did not appear to be playing any role in producing the final behavior, yet success in this task depends on these readings. Clearly any controller that fails to take into account a key environmental characteristic is liable to experience brittle performance and even failure in unusual conditions as evidenced by the poor cyclic behavior that occurred when the light was overstepped. Employing a more intermittent goal may alleviate these problems.

3.2.2 One-step goals

The most common strategy developed by one-step goal networks is much like the constant goal behavior except that the one-step goal controller consistently switches direction as soon as the goal changes and a more reasonable response to overstepping the light is used. Figure 3.9 shows some typical one-step goal behavior. Notice in the instance that the robot overstepped the light, the robot was able to return directly to the light after contacting the boundary. A quick look at the one-step weights will

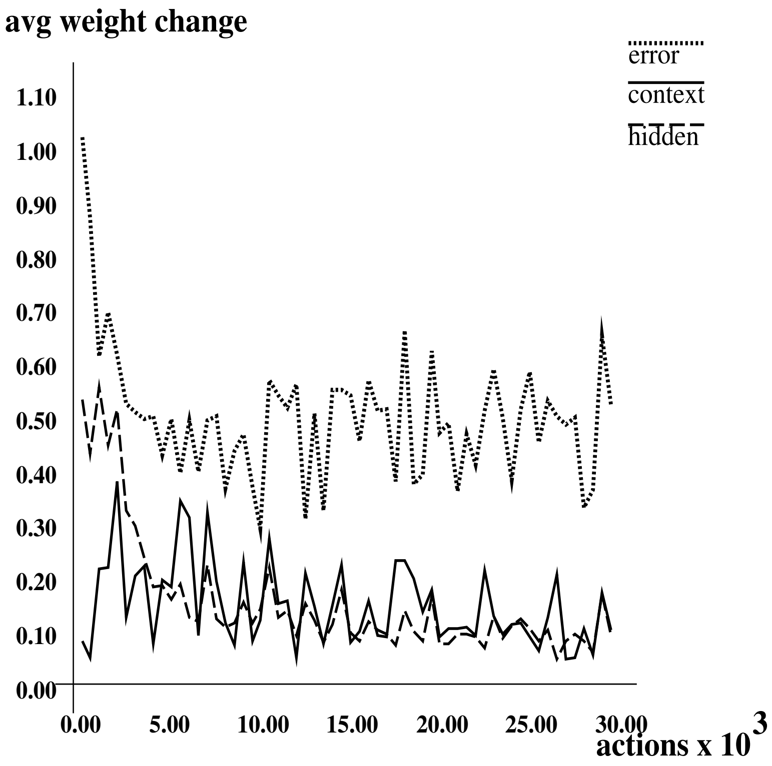

Figure 3.8: Constant goal: weight changes during learning. The topmost curve represents the error during reinforcement learning. The next two curves represent the average magnitude of the weight changes occurring in the weights emanating *from* the context and hidden layers.

reveal why.

Figures 3.10 and 3.11 show the network's weights. Once again, the one-step goal network uses $H0$ to indicate right motion which is associated with avoiding the light and $H1$ for the opposite. The primary differences between the one-step goal weights and the constant goal weights are in the weights from the light sensor and from the context layer. Let's examine how these differences affect the processing in the network. On the initial step of a seek goal, the goal unit will activate $H1$ causing the robot to move left. From that point on, however, the goal unit will be completely inactive (until the light is reached and a new goal is in effect). However, $H1$ is able to sustain high activation, even though it sends itself inhibitory feedback through the context layer, through the positive weight coming from the light sensor. Notice that there is also positive weight from the light sensor to $H0$, but it is not large enough to fully activate $H0$ on its own.

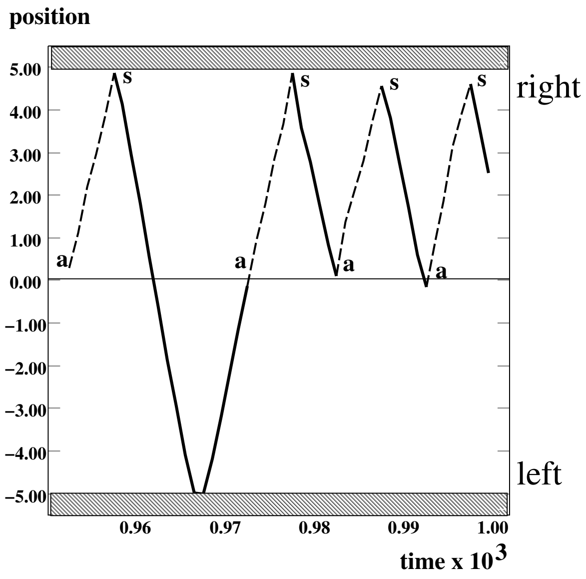

Figure 3.9: One-step goal: behavior

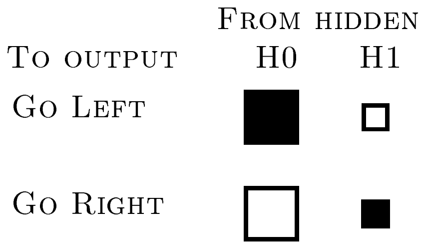

Figure 3.10: One-step goal: hidden to output weights

If the light is overstepped during a seek goal, the one-step controller will continue to move the robot to the left until the boundary is contacted. Then the left sensor will be triggered activating $H0$, but $H1$ will remain active for at least one step causing the robot to not move and leaving it in contact with the boundary. Eventually inhibition from $H0$ will deactivate $H1$ causing the robot to move right towards the light again. Without a constant goal to reactivate $H1$, $H0$ will maintain the rightward motion until the light is reached and thus avoid the ineffective cycling of before.

The graph of the hidden activations for this one-step network, shown in Figure 3.12, reveals that for the typical seek and avoid behaviors the activation of each hidden unit is again correlated with one goal: $H0$ with avoid and $H1$ with seek. Only the instance where the light was overstepped does $H0$ become active for a seek phase

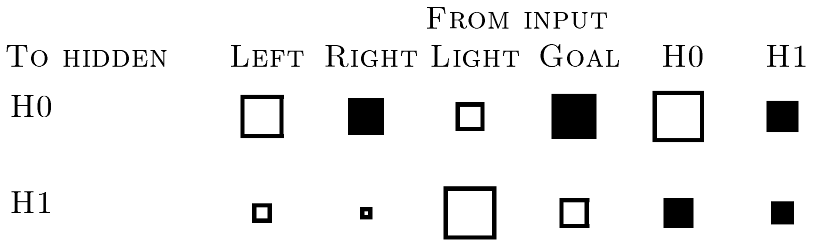

Figure 3.11: One-step goal: input to hidden weights

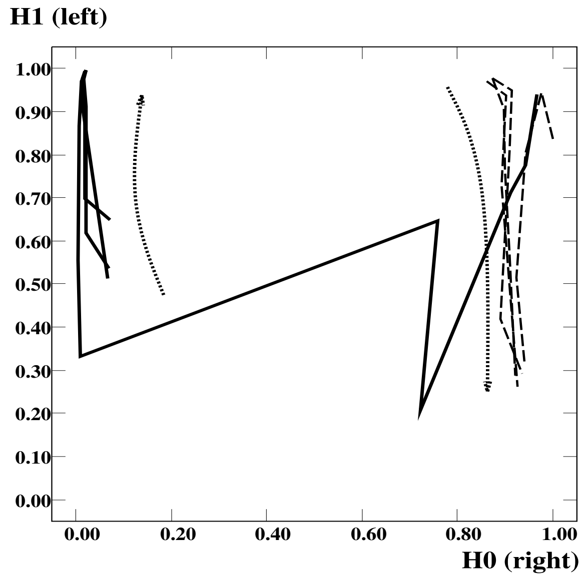

Figure 3.12: One-step: hidden activations through time

of the task.

Figure 3.13 shows a different course of learning for the one-step goal networks when compared with the constant goal networks. Initially there is again a sharp decrease in error, but it is extended over 10,000 steps rather than 5,000 steps as before. In addition there is a peak of learning in the context weights near the end of this steep decline in error. Then after a long plateau of relatively stable error levels, there is another spurt of learning after 20,000 steps which leads to the high performance of this class of one-step goal controllers. Because the goal is not constantly present, in the one-step case, the network first learns about the reactive features of the task and only later associates the goal with a direction of motion. Thus with intermittent goals, the one-step controllers are able to build a strategy more closely tied to the key

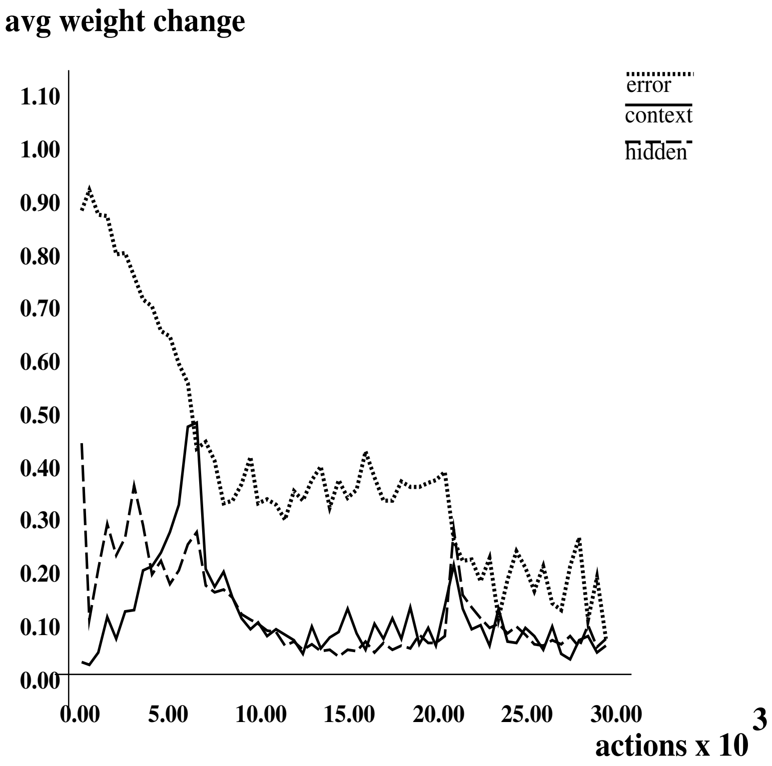

Figure 3.13: One-step goal: weight changes during learning

environmental cues—the light and the sensors—rather than one linked almost solely to the goals.

3.2.3 Implicit goals

The trend in these experiments has been that limiting the presence of goals has improved performance by creating solutions more fine-tuned to the environment; so we might expect that in the absence of goals the controllers would develop the most intricate strategies. Yet, gradient descent learning algorithms like CRBP will tend to settle on the simplest solution that is reasonably successful. And in this implicit goal environment there is very straight-forward strategy that is fairly successful. The robot behaves like a bumper car: it moves one direction until it contacts a boundary and then moves back in the other direction until it contacts the opposite boundary (see Figure 3.14). This behavior will also be adequate whenever the light is overstepped because the robot will eventually head back towards the light.

From Figure 3.15, we can see that once again $H0$ creates right motion and $H1$

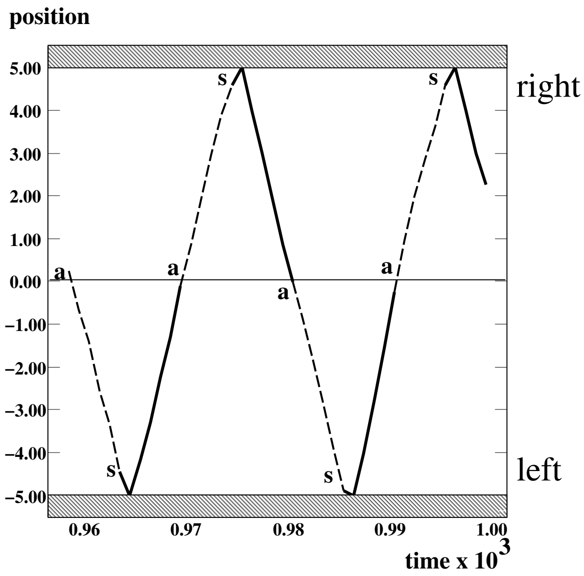

Figure 3.14: Implicit goal: behavior. In these tests the controller is not given any information about the current task goal.

left motion, and based on the outward behavior of the implicit goal controller, we would probably predict that the network has implemented its behavior based only on the contact sensors and not the light sensor. However, in Figure 3.16, it is clear that though $H1$ is initially triggered by the sensors and then sustained by positive feedback through the context layer, $H0$'s activation is dependent on the light. When $H1$ is active it inhibits $H0$ through the context layer, but when $H1$ is turned off by contact with the left boundary, $H0$'s positive bias (not shown) coupled with its excitatory weight from the light sensor turn it on and sustain its activation. When the right boundary is eventually contacted, $H1$ is activated and will then inhibit $H0$ again causing motion in the opposite direction. Thus, the external behavior produced

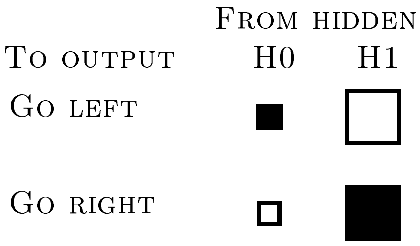

Figure 3.15: Implicit goal: hidden to output weights

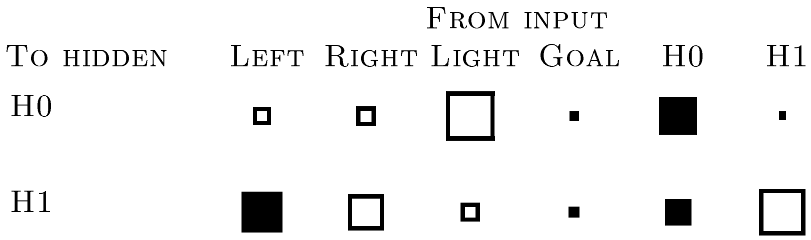

Figure 3.16: Implicit goal: input to hidden weights. Because the activation of the goal unit is always zero, the weights from the goal unit are never modified and remain at their initial small random values.

by a connectionist controller is not always indicative of the internal mechanisms being employed by the network.

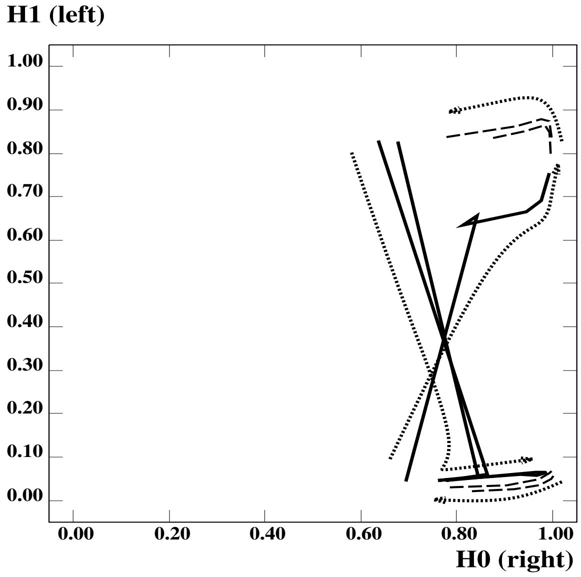

Figure 3.17: Implicit goal: hidden activations through time

Figure 3.17 shows the hidden activations during the implicit goal behavior. The loop in the upward-moving seek traces and the change in direction of the downward-moving seek traces occurred when the robot contacted the boundary and a switch in direction resulted. For the implicit goal variation, the hidden units are not correlated with either goal, but only with a direction of motion.

For the implicit goal networks, the course of the learning is less punctuated (see Figure 3.18). There is a steady decline in error during the first 10,000 steps, yet no

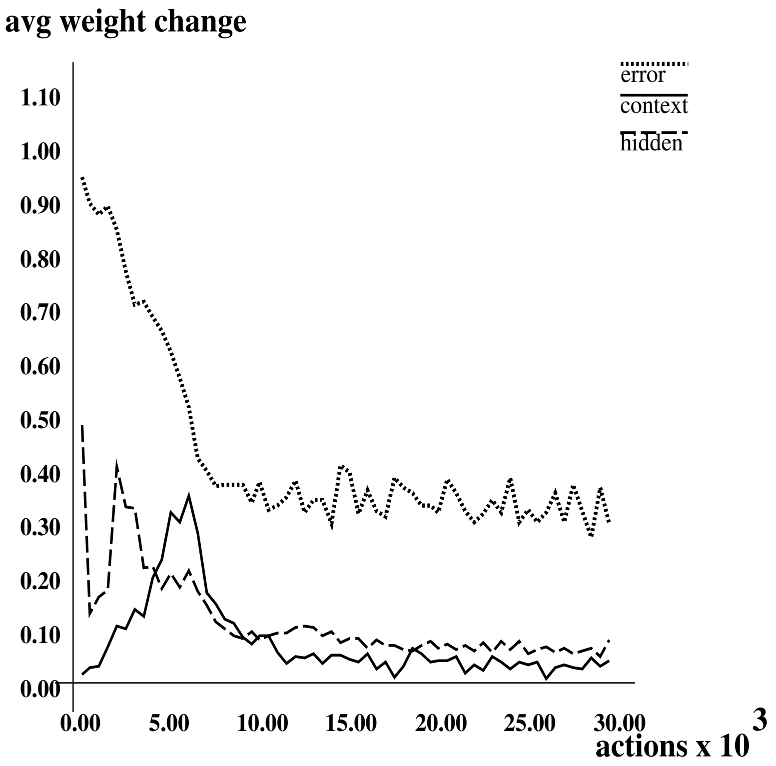

Figure 3.18: Implicit goal: weight changes during learning

further significant learning occurs after this point of convergence. Without explicit goals the network is not driven to improve on its initial strategy.

To summarize the findings to this point, in the simple one-dimensional environment, the most beneficial way to provide goals was intermittently at the initial step of a new task. Networks with access to constant goals failed to learn about important aspects of the environment, instead depending almost solely on the goal. Finally, networks without goals were surprisingly successful despite the lack of guiding information.

3.3 Discontinuous version of the task

Network learning algorithms develop solutions to tasks based on the statistical regularities inherent in the environment. By studying a periodic task in the previous section, where the goals always occurred in sequence (i.e. seek, avoid, seek, avoid, etc.), we may have inadvertently simplified the network's problem of discovering the significance of the goals. How would the results change if the task was modified so that the goals were no longer periodic (and predictable) but were presented in a discontinuous fashion? For example under the discontinuous version, a controller might be faced with the following sequence of goals: seek, seek, seek, avoid, avoid, seek, avoid. To ensure that the robot must move to accomplish these unpredictable goals, its initial position for each new goal will be selected randomly so that it is placed between one and three units from the origin. In addition, the activation values of its short-term memory will be re-initialized to 0.5 prior to attempting a new goal. Thus in the discontinuous form of the task, the goals, the robot's position, and the controller's memory are all discontinuous.

In an attempt to alleviate some of the cyclic behavior seen in the previous set of experiments, a few other modifications to the training process were made. First, the proportion of seek goals to avoid goals was increased. In the previous experiments, the majority of the punishment was received during the seek goal. To improve the network's chances of learning good seek strategies, seek goals were given more frequently (2/3 rather than 1/2 of the time). Second, a limit was placed on the number of actions a controller could spend on one particular goal. If a controller was not able to achieve the given goal within 50 actions then a new goal and position were randomly chosen.

Variation	Avg %pun	SD
Constant	77.6	0.7
One-step	77.0	1.1
Implicit	76.3	1.3

Table 3.3: Baselines: Summary of performance results for the discontinuous version of the task prior to learning

The baseline behavior of the network architecture was tested under the new discontinuous task conditions. Once again, for each variation, ten networks with different initial random weights were used to control the robot for 1,000 actions. The results are shown in Table 3.3. Although learning the discontinuous version of the seekand-avoidlight task is much more difficult, the baseline punishment values don't reflect this. Each of the average punishment values in the discontinuous task baselines is slightly lower than its counterpart in the periodic task baselines (refer back to Table 3.1). This is because in the periodic case, prior to learning, the initial random weights will rarely (if ever) take advantage of the predictable sequence of the goals by chance. But once learning begins, the controllers operating under the periodic task quickly discover this goal regularity and use it to their advantage.

Because the discontinuous version of the task is more difficult than the periodic

Variation	Avg %pun	Best %pun	Worst %pun	SD
Constant	50.7	28.7	59.1	10.3
One-step	38.1	28.3	59.7	10.4
Implicit	45.7	28.9	63.7	12.9

Table 3.4: Summary of performance results for the discontinuous version of the task after learning

version, for the discontinuous set of experiments the training time was extended and the number of trials was halved. For each of the three variations, ten networks with different initial random weights were trained for 100,000 actions using the CRBP algorithm (as opposed to twenty networks and 30,000 actions in the periodic experiments). After training, each network was again tested for 1,000 actions and the average percent of the actions punished, across the ten trials, was used as the performance measure for comparison.

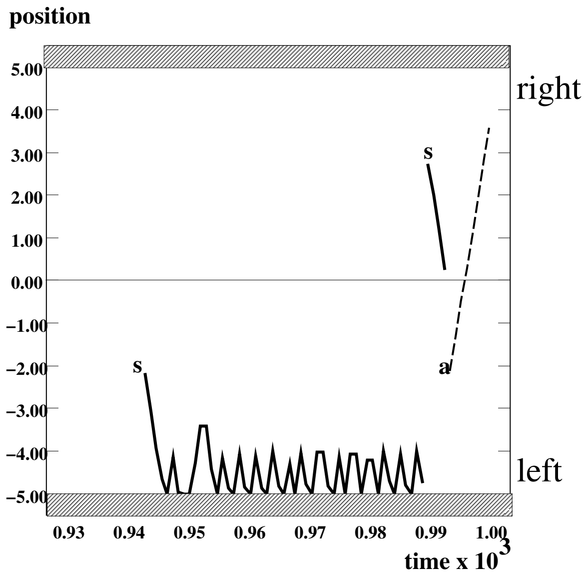

Figure 3.19: Constant goal: discontinuous task behavior still exhibiting cycling

The results for the discontinuous version of the task are shown in Table 3.4. Although the punishments levels are much higher, qualitatively the performance measures for the discontinuous training are quite similar to the previous periodic training: one-step goals are the most effective and constant goals are the least effective with implicit goals falling in between. However in the discontinuous case, a three-way comparison across goal type did not produce significant results. In two-way comparisons, only the constant versus one-step comparison was significant ($p < 0.05$). Note in Table 3.4 that the standard deviations across the goal types for discontinuous training are within 3 units of each other whereas for the periodic training there was a much wider spread of 24 units. By limiting the time allowed to attempt a goal to 50 steps, the effect of poor cycling on the final performance score was diminished.

Despite the qualitative similarities between the results for each type of training, in some cases the behavior produced under discontinuous conditions exhibited some interesting differences from the periodic conditions.

3.3.1 Constant goals

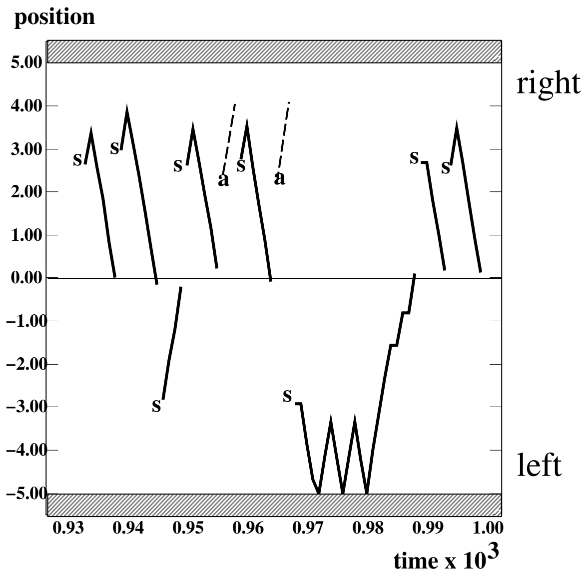

Figure 3.20: Constant goal: discontinuous task behavior with solution to cycling problem

In the constant goal variation, as before most networks ignored the light and focused on the goal, associating one direction with avoiding and the opposite direction with seeking. In Figure 3.19, we see one constant goal controller that produces right actions for avoiding and left actions for seeking regardless of the initial position. During avoid goals, this strategy will initially lead to some punishment if the robot starts on the negative side of the environment, but it will ultimately be a successful strategy. In contrast, for seek goals, this simple strategy is guaranteed to fail half of the time. The ineffective cyclical behavior that in the previous set of experiments occurred relatively infrequently, when the goal was overstepped, is quite prevalent here.

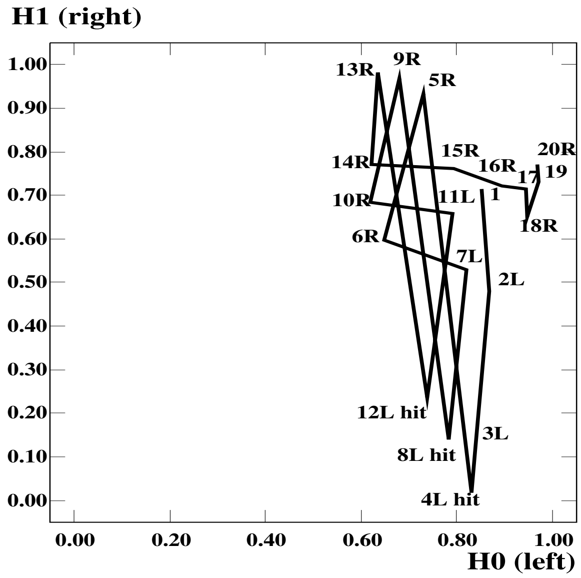

Figure 3.21: Constant goal: hidden activations through time that correspond with the seek behavior from above where a cycle was broken. Points are numbered sequentially and the action taken is indicated. If no action is given, as on points 1, 17, and 19 then the robot was not moving. Contacts with the boundary are noted as hits.

However, there were a few constant goal networks that discovered ways to break out of these cycles and produced successful solutions. For example, see Figure 3.20. For this constant goal controller the tendency was to initially move to the right for any goal, but when the seek goal was in effect the controller could attend to the light gradient and switch directions if necessary to move towards the light.

More importantly, if during a seek goal the robot entered a cycle at one of the boundaries it eventually broke the cycle and achieved the goal within the fifty-step limit. The method for breaking the cycle can be seen through the changes in the hidden activations over time as shown in Figure 3.21. $H0$ is associated with left actions and $H1$ with right actions. At the beginning of this seek sequence, $H0$ is dominant causing the constant goal controller to move the robot left into the boundary. Each time the boundary is contacted, $H1$ becomes momentarily activated, creating several rightward steps. Then $H0$ takes over again and the cycle is created. Notice, however, that each successive time $H1$ becomes activated its residual activation increases. So that by the third contact with the boundary at step 12, it is able to maintain its activation and produce a long enough sequence of right movements to succeed at the seek goal. This result is encouraging, yet only one-fifth of the constant goal networks tested were able to override the ever-present goal and squelch cyclic behavior in this manner.

3.3.2 One-step goals

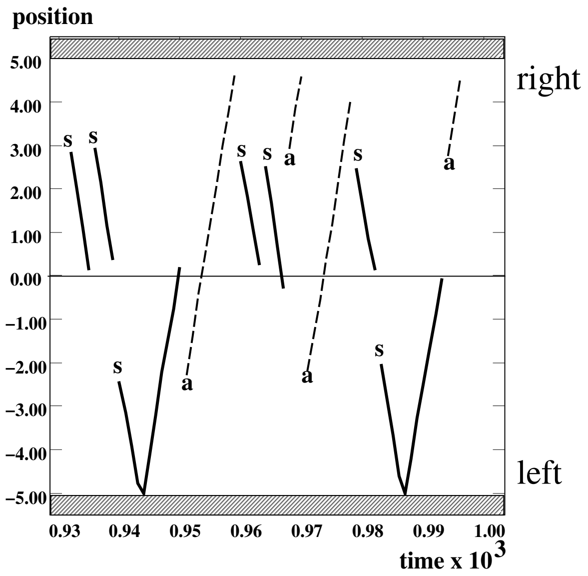

Figure 3.22: One-step goal: discontinuous task behavior

In the one-step goal case, the behavior produced by discontinuous training is the same as for periodic training. The avoid goal is associated with right movements and the seek goal with left movements (see Figure 3.22). If a seek goal is not achieved by left movements, then when the left boundary is contacted the implicit goal controller moves the robot right, directly back to the light without any cycling.

3.3.3 Implicit goals

For the implicit goal case, there is an interesting difference in the learned behavior between training types. Under discontinuous training the most common strategy used by the implicit goal controllers was to head for the light in every new situation. In Figure 3.23, we see that the implicit goal controller initially guesses that the light is to the right. If this is not the case, then the implicit goal controller switches the robot's direction before contacting the nearest boundary. Using this method, the boundaries will almost never be contacted (unless the light is overstepped) and the harder seek goal is achieved quite efficiently.

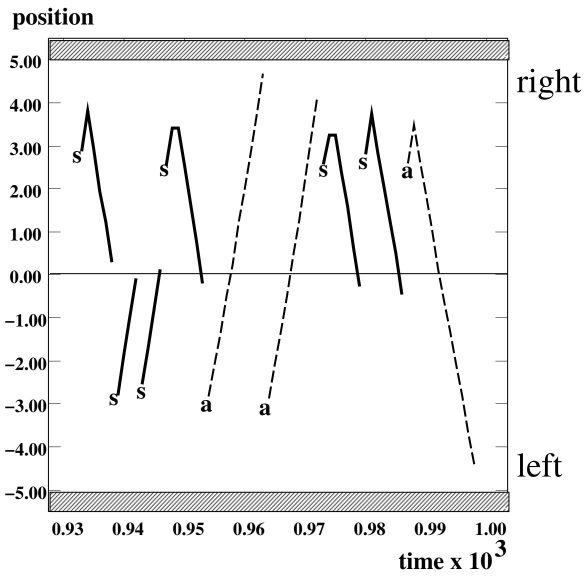

Figure 3.23: Implicit goal: discontinuous task behavior

For implicit goal controllers, the primary source of punishment is following the light gradient incorrectly during avoid goals. This experimental variation was the only case where more punishment was received during avoid goals than during seek goals. Recall though, that the implicit goal controller is not given any information about which goal is currently in effect and this information is not predictable from the context. Lacking goal information, the implicit goal controllers trained under discontinuous conditions become finely tuned to the changes in the light readings. In Figure 3.24, we can see that one of the hidden units has learned to closely mimic the actual light readings. In this way, the output layer has direct access to information about the light gradient and can use it to determine appropriate actions.

3.4 Discussion

In this chapter, the effect of abstract goals on the development of network controllers for a simple one-dimensional robot has been examined. Varying the persistence of goals from continuous, to intermittent, to absent, has revealed that the inclusion of goals does significantly change the performance of the controller, as well as the pattern of the weights, the use of the context memory, and the course of learning. The experiments presented in this chapter have demonstrated that the connectionist approach to control truly allows the task demands to be the primary force in shaping

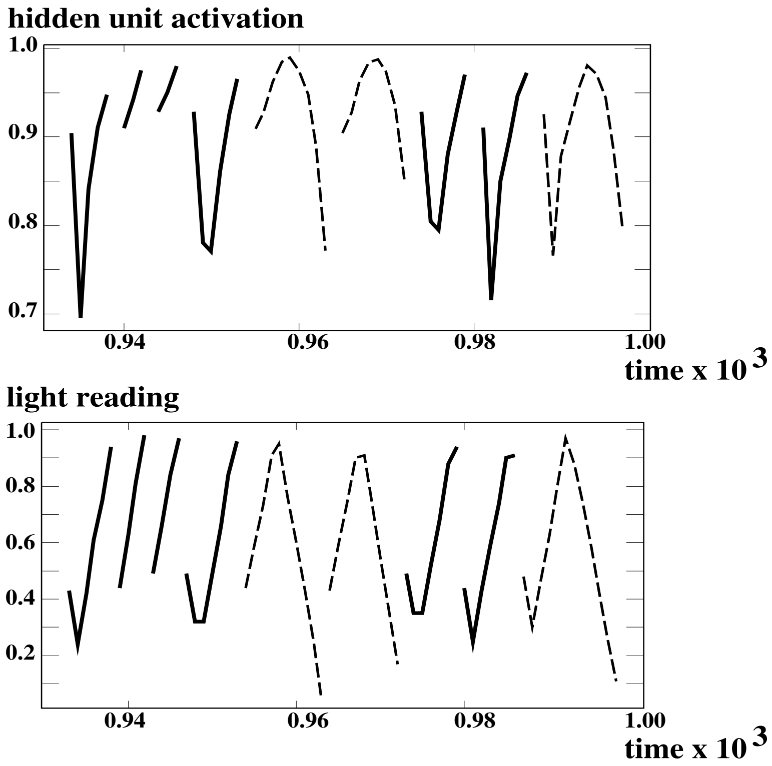

Figure 3.24: Implicit goal: activations of one hidden unit closely tracks the actual light readings over time for the behavior shown in the previous figure.

behavior.

When constant goals are present, the control networks learn to depend almost solely on them to guide behavior, leading to early convergence and poor generalization. When the goals are implicit, the control networks develop a single global strategy that is successful for both aspects of the task, but further learning is limited by the lack of task information. The most successful control networks employed one-step goals. This intermittent presentation of goal information allowed the networks to form crucial associations about the environmental features and still incorporate the external guidance when it was available.

Two opposing styles of training were also compared. During periodic training, the seek and avoid goals always occurred in sequence. To ensure that this predictable pattern of goals was not a key factor in the results, an additional set of experiments was performed using a discontinuous goal presentation. Although the results were qualitatively the same in many respects, the harder discontinuous task led to the development of more complex strategies for the constant and implicit goal variations. In a few constant goal controllers, a method for breaking cyclic behavior was discovered and in many implicit goal controllers the light gradient was used to avoid contacting the boundaries.

Certainly these experiments are only a start in reconsidering the concept of goals for AI; however, the type of abstract goals examined here are more similar to the sort of real-world goals we typically encounter as learners than symbolic state descriptions. Initially a teacher's instructions may be largely uninterpretable, but through observation of the context in which they occur, their relevance gradually emerges. Eventually an understanding of the goals is gained and appropriate action can be taken in response to them.

As demonstrated in this chapter, applying connectionism to robotics problems offers a good testbed for exploring other possible goal forms. For instance another possibility that could be examined here is that goals could decay over time, eventually disappearing. Controllers trained in this way might develop self-regulating strategies. Perhaps the most interesting lesson of these goal experiments is that goals need not be symbolic descriptions of a desired state to be effective. Instead, a goal can be viewed more as a trigger whose effect must be properly blended with the other sensations during learning to produce robust behavior.

4

Comparing local and global reinforcement methods

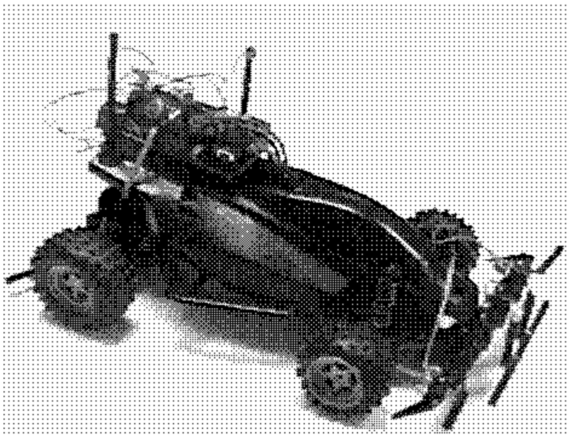

Figure 4.1: A digitized photograph of carbot.

In this chapter, the two reinforcement methods used for adapting connectionist robot controllers, CRBP and GAs, are contrasted. These methods are applied to a real robot, called carbot, shown in Figure 4.1. A number of experiments are described where the presence of goals and the immediacy of the reinforcement are varied.

Although the task used here is essentially an expansion to more dimensions of the task used earlier in the one-dimensional world, it is worth investigating the effects of goals again now that a more realistic environment is being employed. Previously we saw that constantly present goals caused developing networks to prematurely converge on strategies that depended on the goals, and to ignore key environmental features. However, if the expanded environment is sufficiently complex, then we

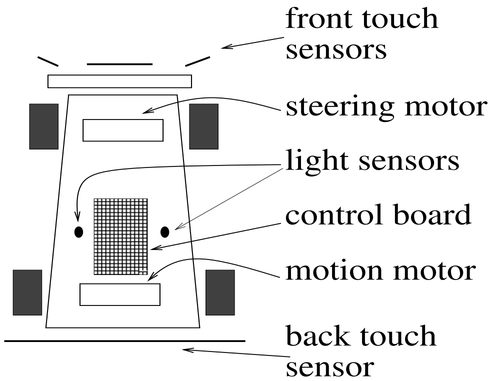

Figure 4.2: Positioning of sensors, motors, and control board on carbot.

would expect that the constant goals may no longer be the most easily associable feature of the environment, allowing the networks to initially attend to the perceptual characteristics.

These experiments are not meant to determine which adaptation method is the best for learning to control robots. Rather the intent is to examine the control behaviors produced by each method and to discover how the algorithms' strengths and weaknesses are revealed by varying the presence of the goal and the immediacy of feedback.

4.1 Carbot—an autonomous robot

4.1.1 The vehicle

Carbot is a modified toy car (6 inches wide, 9 inches long, and 4 inches high) controlled by a programmable mini-board (designed at MIT by Fred Martin (Martin, 1992)). This board allows remote or on-board control, although the on-board memory is quite small. Because of this carbot was tethered to a PC during execution. Carbot was inexpensive to build, primarily because it makes use of primitive sensors—no lasers, video, or sonar. The robot has just two types of sensors: digital touch sensors on the front and back bumpers, and analog light sensors on stalks near the back which are directed 30 degrees to each side. It has two servo-motors; one controls forward and backward motion and the other steering. See Figure 4.2 for a schematic drawing of carbot that more clearly shows the position of the sensors and motors.

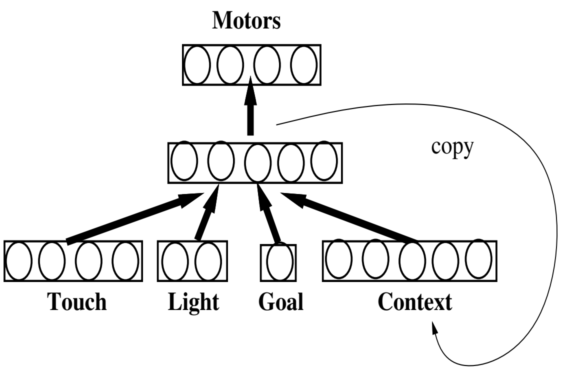

Figure 4.3: Carbot's control network.

4.1.2 The task

The environment, called the *playpen*, is a rectangular box (2 feet by 4 feet) with a light in one corner. Carbot's task within this environment is just as before. At the reactive level it must perform rudimentary navigation by avoiding walls and continually moving. See (Meeden et al., 1993) for a detailed description of factors that affect a network's performance when learning this type of simple reactive task.

At the goal level, carbot must again either seek or avoid the light depending on the current goal. The goal varied in the periodic fashion—seek always followed avoid. Discontinuous training will not be examined here.

4.1.3 The control network

Carbot is controlled by a remote connectionist network that communicates with the mini-board. The network gathers input data from the sensors and determines how to set the motors for the next time step. Figure 4.3 shows the standard network used to control carbot for the experiments described in this chapter.

There are four sets of input units; two sets are for sensors, one is for goals, and the last is for context memory. The first set of units in the input layer represents the state of the digital touch sensors. There are four digital sensors—three in front and one in back. Two of the front sensors are out to either side so that carbot can sense side collisions when moving forward. When the digital sensors are triggered by contact they take on the value one, otherwise they are zero. The next set of two units represent the state of the analog light sensors, whose values can range from 0.0 to 1.0 (due to ambient light, their values rarely fall below 0.25). The next set of one unit encodes the network's goals. Again only two goals are used: +1 for seeking the light and -1 for avoiding the light. Recall that the last set of units, called the context, provides a limited short-term memory of the network's past states. Activations from the hidden layer on the previous time step are copied into the context units directly. For this set of experiments the size of this short-term memory has been increased to five units.

The four output units of the network determine what the activity of the motors will be for the next time step. Carbot is propelled by two motors each requiring two units to specify its state. The first output unit represents the spin direction of the rear motor (this determines direction of motion—forward or backward). The second unit designates the state of the motor as on or off. The third unit represents the spin direction of the front motor (this determines the direction of turning—left or right). The fourth unit designates the state of the front motor as on or off. In order to turn, carbot must have both motors running—the back motor provides movement while the front motor steers.

4.1.4 Real-world versus simulation training

Since carbot has to physically move in the world, bumping into things, it takes a long time to train and test a particular controller network. Each real-world action is executed for one second, so 5,000 actions require approximately 2.5 hours to be completed. To alleviate this time limitation, a software simulation was implemented.

Although simulators are often too perfect and not very realistic, when the simulator is based on an actual robot, the simulator's behavior can be programmed to closely correspond with the real-world behavior. Harvey suggests some ways to ensure that a simulator stays in close step with reality:

1. Simulations of the inputs to sensors and the outputs to actuators should be based on carefully collected empirical data.

2. Noise must be taken into account at all levels.

3. The simulation can be calibrated by testing adapted architectures in the real robot.

4. A range of unstructured, dynamic environments should be used to ensure robustness.
(Harvey, 1993b)

In constructing carbot's simulator, Harvey's first three suggestions were followed (with respect to the fourth suggestion of employing a range of environments, only the playpen was used although a number of different configurations were tried). As per the first suggestion, carbot's sensor readings, average turning radius and distance traveled for each action in the real world were empirically determined and used as the basis for the simulator. On average, each straight motion propels carbot seven inches and each turning motion results in a linear change in position of four and a half inches. Each turn typically changes carbot's heading by twenty-four degrees. With respect to the second suggestion, small amounts of random noise were added to carbot's heading, position, and light sensor readings after each action taken within the simulator.

For the third suggestion, a number of experiments were done to verify that the sorts of controllers that worked well in the simulator would also work well in the real robot. Controller networks trained in the simulator were transplanted to the actual robot and tested without further training. The behaviors produced by the same network on the simulator and on the robot were compared in terms of percent time punished. These transplant tests provided a surprising result: the robot's performance was always superior to the simulator's performance. This result is probably due to the additional noise provided in the simulator. The actual robot's movements are only occasionally noisy while the simulator's movements are systematically noisy and this added noise apparently enhances learning by exposing the controller to a wider range of environmental conditions. I hope to quantify the benefits of added noise in future experiments.

Thus the simulated robot's behavior was determined to be close enough to the actual behavior of carbot to warrant use of the simulator for research. The simulator is used to test hypotheses and to develop useful architectures that are then applied back to the real robot. This saves a significant amount of time since 5,000 simulated actions can be executed in less than a minute (a speedup factor of 150).

4.2 Determining carbot's fitness in the genetic algorithm

In the CRBP algorithm all types of punishment are treated equally. Regardless of whether the punishment came from not moving, bumping into a wall, or not following the light gradient correctly, the consequences are the same. Similarly all types of reward are treated equally. Recall though, that rewarded actions are reinforced with a higher learning rate than punished actions. It would be possible to differentiate between types of punishment or types of reward by further varying the learning rate by type. For example, following the light gradient correctly could result in a lower learning rate than actually achieving the final light goal. This type of variation has been tried, but the results were not significantly different than those obtained using the standard (and simpler) method.

Unlike the case for CRBP, it is more straightforward to differentiate between various types of reward and punishment in the GA because each type of reinforcement can be assigned a unique feedback value. Here's how carbot's actions were reinforced during GA processing:

- Accomplished a light goal: $+50$
- Any touch sensor triggered: -7
- Not moving: -2
- Not following light gradient correctly for goal: -1
- Following light gradient correctly for goal: $+5$

Each action received only one of these possible values and the evaluation of an action was checked in the order given above. So if a goal was accomplished by an action that also caused carbot to hit a wall, the reinforcement for that action would be $+50$ and not -7 or the combination of the two, $+43$.

Note that these feedback values represent my bias about the task. By making the punishment for hitting a wall more than three times greater than the punishment for not moving, the GA will tend to focus effort on avoiding walls first. However, the GA is not highly sensitive to the ratios of these values and only a limited amount of trial and error was needed to choose this set of values.

Based on these feedback values we can determine the range of possible fitness values. Recall that the average reinforcement received over two sequences of 50 actions is used to determine a network's fitness. To calculate the minimum fitness value, assume that every action results in the minimum feedback value (-7 for hitting a wall).

$min(fitness) = actions \times -7 = -350$

To calculate the maximum fitness value, assume that an optimal strategy for seeking or avoiding the light requires only five actions and is never punished. In my observations of carbot, the light goals can occasionally be accomplished in only four actions, but five seems to be the best average. So a fifth of the actions will receive the accomplished goal feedback of $+50$ while the remaining actions will receive a reward of $+1$

for correctly following the light gradient.

$max(fitness) = (0.2actions \times 50) + (0.8actions \times 1) = 700$

In the experiments described below, the genetic algorithm works with a population of 250 networks. Each network is initialized with small random weights and then its fitness is measured as just described. Typically this initial population has an average fitness of approximately -100 where the worst individual has a fitness of about -230 and the best a fitness of about $+50$. After training, these measures have improved to approximately $+185$ for the average fitness, -45 for the worst individual, and $+400$ for the best individual.

Individual genes in the GA for this carbot domain have a length of 89—the networks each have 80 weights and 9 biases. Referring back to Figure 4.3, recall that the layers are fully connected. There are $12 \times 5 = 60$ weights from the input to hidden layer and $5 \times 4 = 20$ weights from the hidden to output layer. Only units in the hidden and output layers have biases: $5 + 4 = 9$.

4.3 Varying goals and the immediacy of feedback

For each type of experiment described below, three trials were done with a GA and five trials were done with CRBP. Fewer trials were done using the GA method because of the amount of processing time needed. All the experiments were performed in the simulator.

Four types of experiments were performed: three varied the goals from constant, to one-step, to implicit while using immediate feedback (as done in the previous chapter), and one used constant goals with delayed feedback. Each GA run consisted of 1000 generations. The number of cycles performed for each CRBP run depended on the type of experiment: 200,000 cycles for the constant goals; 300,000 cycles for one-step and implicit goals; and 500,000 cycles for the constant goals with delayed feedback.

In the first set of experiments, the adaptation methods had access to the most information—both constantly present goals and immediate feedback about the light gradient were provided. On the input layer an explicit goal was given: the sign of the activation value determined which mode the controller should be in (either seek or avoid). This goal is explicit in the sense that it is unambiguous, however the controller must learn what this goal means and how to react based on its value. The light gradient feedback was given through the reinforcement procedure. When in seek-mode, a controller was rewarded if the sum of carbot's light sensor readings increased relative to the previous time step; in avoid-mode, a controller was rewarded if the sum of the readings decreased relative to the previous time step.

In the second set of experiments, the one-step case, the goal was present for the single time step after the previous goal had been accomplished (to indicate the new goal). After that point no goal information was given, but immediate feedback about the light gradient was always provided.

In the third set of experiments, the implicit goal case, the goal unit remained in the input layer but its activation was always zero. Feedback about the light gradient was still provided. By varying the presence of a trigger-like goal we can investigate whether one-step goals are still the most advantageous in carbot's more complex domain.

In the final set of experiments, a constant goal was returned and the light gradient feedback was removed. Under this variation, CRBP receives delayed information about the light, but still receives immediate reinforcement about contacting walls and not moving. Although the GA is already considered a delayed reinforcement procedure, for this set of experiments all feedback about the light was removed except for the +50 feedback when a goal was achieved. By varying the immediacy of the reinforcement we can investigate whether the CRBP can adequately learn the task with only delayed feedback and whether the GA can learn the task without the gradient information.

4.3.1 Evaluating behavior

To evaluate and compare each control network's performance, 100 random situations were set aside for generalization testing. A situation consists of a goal, an (x, y) position, and a heading. There are approximately 8×10^5 possible situations if the environment is discretized into one-inch units and the heading into one-degree units (2 goals, 48 x positions, 24 y positions, and 360 headings).

After training, each controller network was tested for 50 actions over each of these same 100 randomly selected situations. The average percent of actions punished over all these situations was used as the performance measure for each controller network.

All of the statistical analyses reported below make use of analysis of variance (ANOVA) testing. Several types of comparisons were done: across an adaptation type: (constant goal versus one-step goal, constant goal versus implicit goal, immediate versus delayed feedback) and between adaptation types (CRBP versus GA). Unless specifically noted, differences were not significant.

Some significant results may have been obscured by the small number of trials

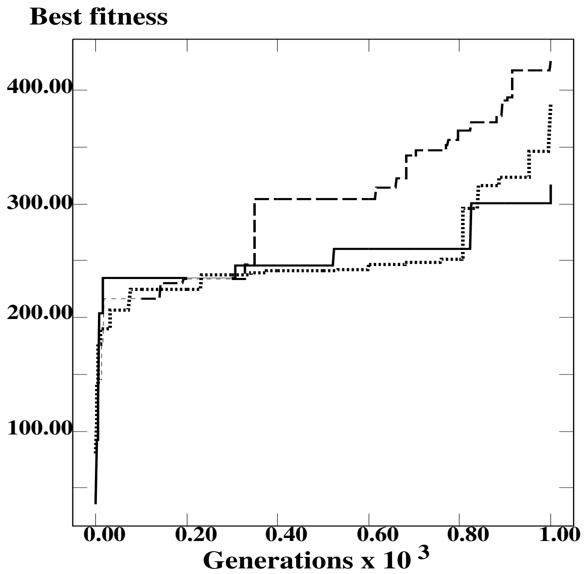

Figure 4.4: Different speeds of convergence in three trials of a GA.

performed and the high variance across the performance metric. For CRBP, this variance is largely a result of the goodness of the initial random weights, whereas for the GA, it is due to differing speeds of convergence. Figure 4.4 shows the fitness of the best individual in the population by generation for the three different trials of the GA constant goal experiments. None of these trials has converged on a final solution since the fitness is still climbing, and one trial is significantly slower than other two. Both types of variability can be reduced if tough performance criteria are imposed to determine when to stop the learning (Whitley et al., 1993). In these experiments, learning time was based on the number of actions performed rather than a particular performance standard. Despite the fact that the performance levels were often not significantly different statistically, as will be seen below, the learned strategies across variations are clearly quite different in character.

Variation	Avg %pun	SD
Constant	85.0	5.9
One-step	82.8	5.1
Implicit	77.8	5.3

Table 4.1: Carbot baselines: Summary of performance results for experimental variations prior to learning

As a baseline for measuring the effect of the adaptation methods, the performance of the networks prior to any learning was examined. Five networks with the architecture of Figure 4.3 were initialized with random weights. Then they were used to control the simulated carbot for 50 actions from the 100 random situations. The results are shown in Table 4.1. The baseline behavior for the delayed feedback experiments is not shown because it is the same as for the constant goal experiments. The only change between these two variations occurs in how the learning is executed and not in the input values provided.

	CRBP		GA	
Variation	Avg %pun	SD	Avg %pun	SD
Constant	41.2	8.7	48.0	7.8
One-step	62.6	5.9	46.3	2.1
Implicit	56.2	4.1	48.7	4.2
Delay	69.2	1.6	52.0	5.3

Table 4.2: Summary of generalization performance by adaptation type.

Table 4.2 summarizes the performance results after learning. Once again learning has significantly improved performance, relative to the baselines, in all the experimental variations. Perhaps the most noticeable feature of these results is that the performance of the GA-trained controllers is quite stable across the variations while the performance of the CRBP-trained controllers is not. More importantly, the CRBP results for the goal variations using carbot contradict the findings from the experiments in the one-dimensional simulated world. In carbot's more realistic environment, constant goals are the most beneficial while one-step goals are the least beneficial—the complement of the previous results. There appears to be an interaction between the complexity of the environment and appropriate availability of goals.

How much more complex is carbot's world when compared to the one-dimensional world? Carbot can execute seven different actions (`forward`, `forward left`, `forward right`, `backward`, `backward left`, `backward right`, and `not moving`) while the one-dimensional robot had only three options (`left`, `right`, and `not moving`). Carbot has four touch sensors and two light sensors where the one-dimensional robot had two touch sensors and one light sensor. In addition, the extent of carbot's domain is significantly larger. The increased number of sensors coupled with the much larger domain leads to a steep increase in the number of possible states that carbot can face. The developing controller network must learn a mapping from states to actions, and in carbot's world this will be a much more difficult assignment. The interaction between the complexity of carbot's environment and the appropriate availability of goals will be discussed further in the detailed descriptions of the experiments to follow.

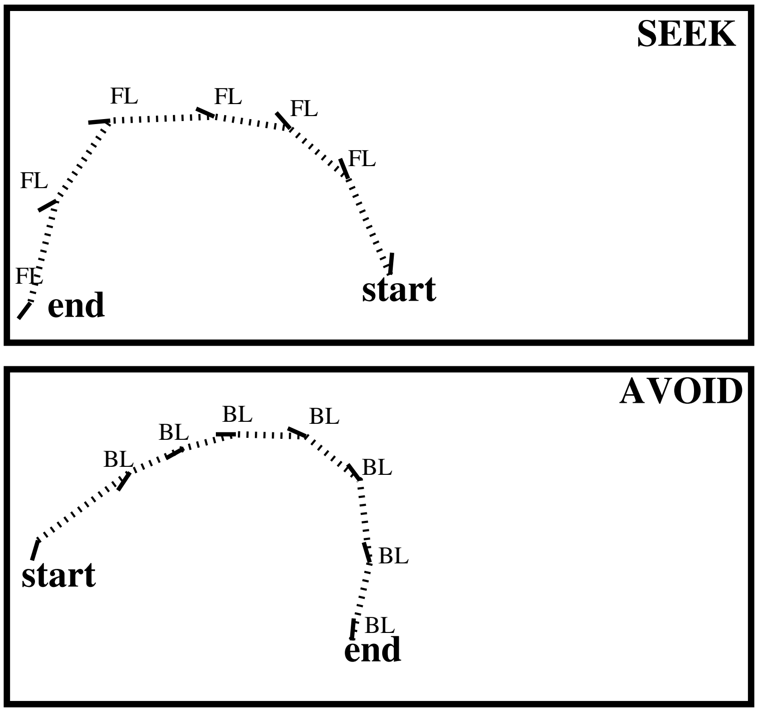

Figure 4.5: CRBP constant goal: Forward semi-circles for seeking the light; backward semi-circles for avoiding the light.

4.3.2 Constant goals with immediate feedback

For CRBP-trained constant goal controllers, the most frequently developed strategy for solving the seek and avoid the light task was to use a forward semi-circle to move toward the light and a backward semi-circle to move away from the light. Figure 4.5 shows one example of this strategy. The playpen is depicted from a birdseye view where the light is positioned at the lower-left corner. Carbot's path is shown as a dashed curve with the starting and ending locations indicated. Carbot's position over time along this path is indicated by a bold bar whose angle represents the current heading. Next to each bar is a short-hand description of the action taken to reach that position (where FL means forward-left, BL means backward-left, etc). As can be seen in Figure 4.5, the semi-circle strategy effectively follows the light gradient. When seeking the light, successive forward-left actions gradually adjust carbot's heading toward the light. Similarly when avoiding the light, successive backward-left actions have the opposite effect.

Rather than employing alternating semi-circles, the GA networks trained under the same constant goal conditions developed a single full-circle strategy for both goals.

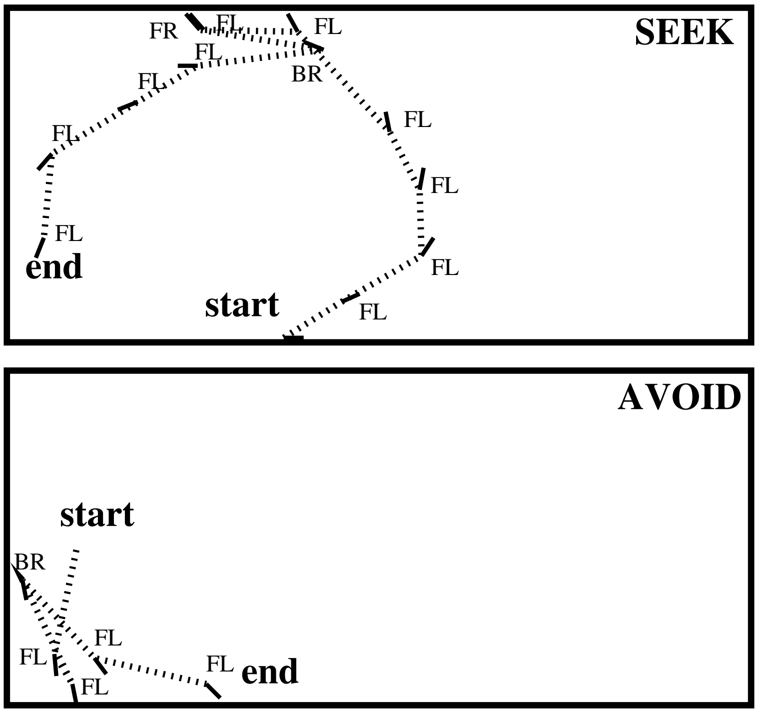

Figure 4.6: GA constant goal: Forward-left circles for both goals. Uses backward-right to move away from walls.

Figure 4.6 shows one complete circle; notice that in this strategy the avoid behavior requires many fewer actions than the seek behavior. Also, because carbot's turning radius is quite large, using a full-circle strategy results in more contacts with the walls of the playpen than does a semi-circle strategy. On average for the constant goal experiments, the GA-trained networks were punished 48.0% of the time while the CRBP-trained networks were punished 44.4%.

One problem with the GA-trained controllers is that they occasionally fall into ineffective cyclical behavior. Figure 4.7 shows one seek case using the full-circle strategy that required 65 actions to achieve the goal. This type of cyclical behavior is rarely seen in the CRBP-trained controllers, probably because their training is continuous rather than being evaluated in relatively short courses of actions as is done in the GA.

For instance, suppose a particular individual in the GA population is an instantiation of the full-circle strategy. If the two random situations picked for determining the fitness are in the front half of the playpen (near the light), then this individual's strategy will be quite successful, yielding a high fitness. However if the two random

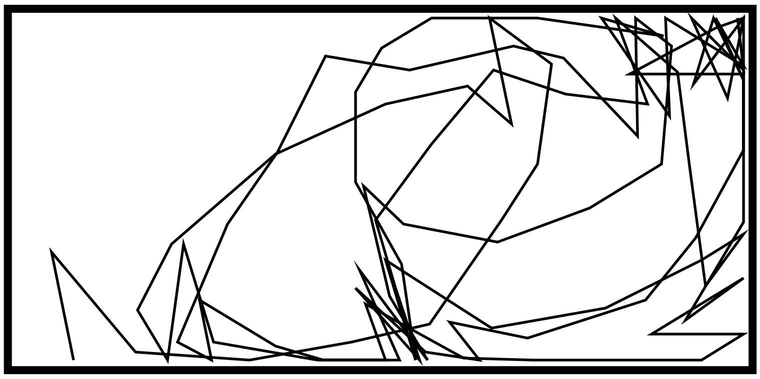

Figure 4.7: GA constant goal: Poor performance using full-circle strategy to seek the light. Carbot's heading and the action descriptions have been omitted to make the cyclical behavior more evident.

situations are in the back half of the playpen this individual's strategy may turn out to be quite poor. But as long as this strategy is sometimes very good it will continue to be explored by the GA. This problem could probably be alleviated by basing the fitness on more experience in the environment. For CRBP's periodic training, carbot's behavior is part of one long connected course of action and thus will have to be more consistently effective.

In contrast to the findings from the one-dimensional simulations where constantly present goals often led to ineffective cyclic behavior, in carbot's domain, constant goals actually enhance CRBP-based learning. Recall that the one-dimensional controllers developed strategies that relied too much on the goals. Carbot controllers are able to use the constant goals without completely depending on them. An interesting aspect of the kinds of strategies developed by the two adaptation methods is that CRBP-trained constant goal controllers pay more attention to the goal than GA-trained constant goal controllers. In every case, CRBP's solutions use different strategies for the two goals while the GA's solutions use the same global strategy throughout the task. To investigate this observation further the magnitude of the weights from the goal unit to the hidden layer were compared to the magnitude of the rest of the weights for each method's solutions. For the CRBP-trained constant goal networks, the ratio of the average goal weight magnitude to the average of the other weight magnitudes was 0.99, indicating that the goal weights are approximately equal in magnitude to the rest of the weights. For the GA-trained constant goal networks this ratio was 0.70, providing some evidence that the goal may not be as integral to the GA solutions. From this result we should expect that the GA's performance in the next set of experiments, where the goal's presence is reduced to a single step will not diminish as much as CRBP's performance.

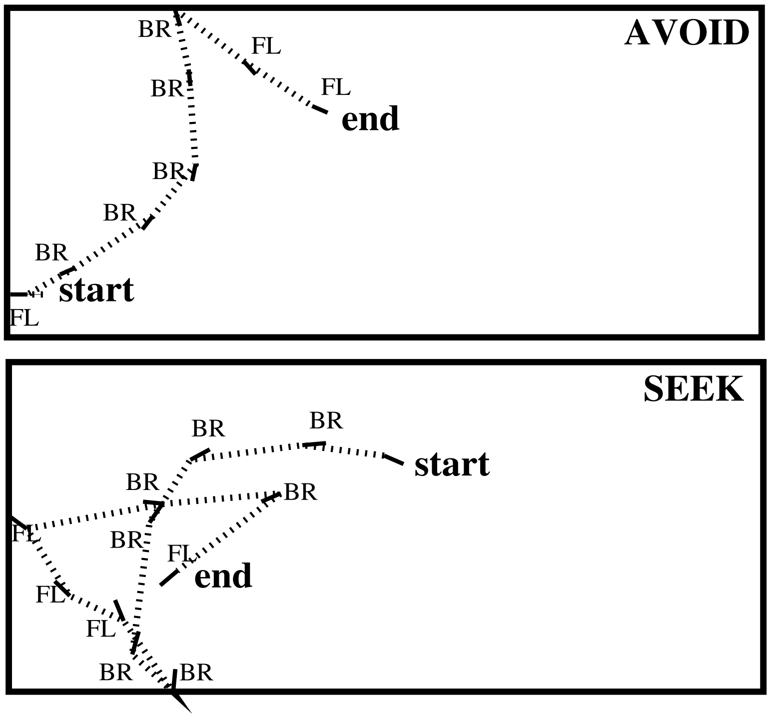

Figure 4.8: GA one-step goal: Alternating series of backward-right and forward-left actions actions.

4.3.3 One-step goals with immediate feedback

For CRBP-trained carbot networks, one-step goals are the least successful form of goal, even worse than no goal at all (as will be seen in the next section). Additional training did not improve the performance level. Although the learning had converged, the networks had not developed a discernible strategy, receiving punishment 62.6% of the time, significantly worse than the 41.2% for the constant goals ($p < 0.01$).

For CRBP-trained carbot networks, the one-step goal occurs so infrequently that it is not informative. In carbot's environment (as opposed to the one-dimensional world), it is much more difficult to fortuitously happen upon the light goals. In the initial stages of learning it may take hundreds of actions to achieve the light goals. When an input goal is only present at the onset of such a long series of actions, it will be quite difficult, if not impossible, to associate its presence with any perceptual regularities. Indeed, the one-step goal inhibits the learning process by creating noise, since the performance improves when the goal is completely removed in the implicit case.

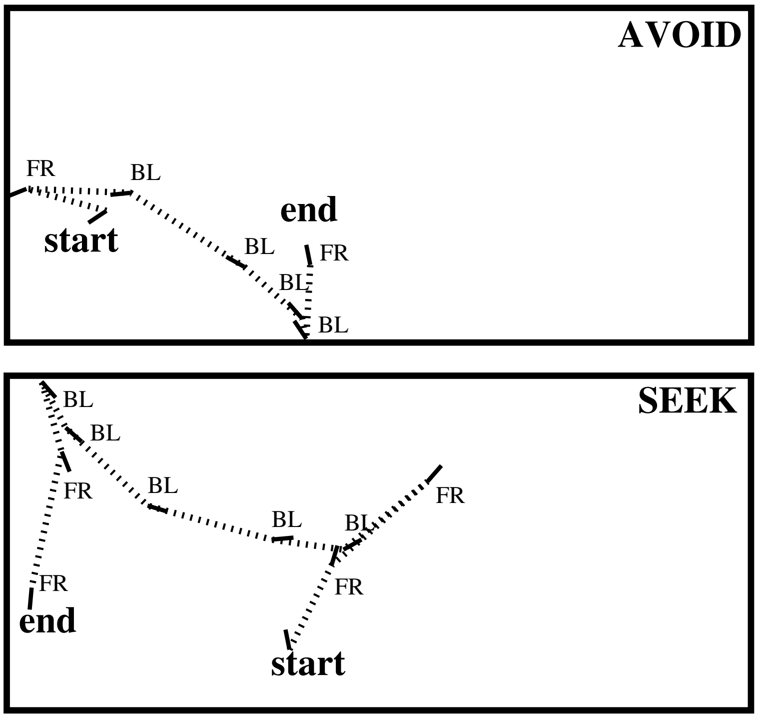

Figure 4.9: CRBP implicit goal (same strategy also found by GA): Alternating series of forward-right and backward-left actions. Executes a two-point turn for both goals.

As predicted, the GA-trained networks are not affected by diminishing the presence of the goal. In fact the average performance is slightly better for the one-step goal versus the constant goal case. Figure 4.8 shows the typical one-step goal strategy developed by the GA. This behavior is quite similar to the two-point turn strategy that the GA developed for the implicit goal case (see Figure 4.9 in the next section). For this experimental variation, the GA's performance of 46.3% punishment was significantly better than CRBP's performance of 62.6% punishment ($p < 0.01$).

4.3.4 Implicit goals with immediate feedback

For this set of implicit goal experiments, CRBP and the GA actually did produce similar behavioral strategies in some instances. The most prevalent CRBP implicit goal solution, which was also found by the GA, was to employ a two-point turn for both seeking and avoiding the light. This was implemented as a short series of forward-right actions, followed by a longer series of backward-left actions, and then another short series of forward-right actions (see Figure 4.9). The switch between

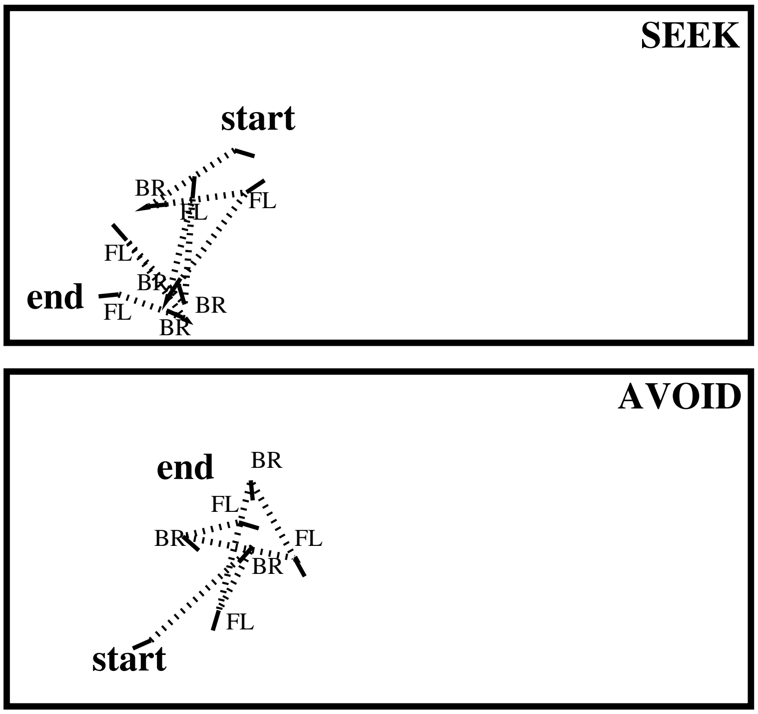

Figure 4.10: GA implicit goal: Alternating backward-right and forward-left actions for both goals. The resulting pattern sometimes looks like a star.

turning directions was usually triggered by a wall, since the playpen is so small, but in some cases the switch occurred without this environmental cue, as is seen at the start of the seek phase in Figure 4.9. Notice that in the initial steps of this strategy the light gradient is actually ignored—to seek the light the controllers begin by moving carbot further away from the light, decreasing the light sensor readings—but once the backward-left portion of the strategy is entered, the sum of light readings steadily increases as carbot's heading gradually turns more towards the light.

The GA-trained controllers discovered another implicit goal strategy as well—a star pattern or a six-point turn—which is shown in Figure 4.10. This behavior is implemented by alternating between single backward-right and forward-left actions rather than a series of turns as was used for the two-point behavior described above. As was the case with the GA's full-circle strategy, the GA's star strategy can be quite ineffective if carbot's initial position is far from the light. For example, in Figure 4.11, 300 actions were executed and the seek goal was still not achieved.

On average for the implicit goal experiments, the CRBP-trained networks were punished 56.2% of the time which is significantly worse than the performance of the

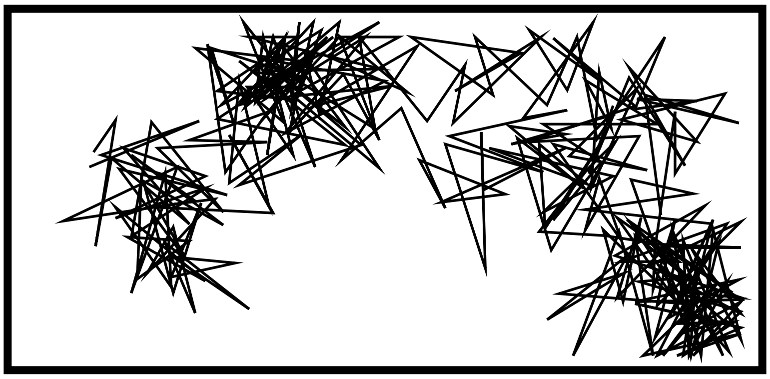

Figure 4.11: GA implicit goal: Poor performance using the star strategy. Carbot's heading and the action descriptions have been omitted to make the cyclical behavior more evident.

CRBP-trained constant goal networks at 41.2% ($p < 0.05$). Clearly the CRBP-trained networks can learn a more effective behavioral strategy when provided with explicit goals which are consistently available. The GA-trained networks were punished only 48.7% of the time which is not significantly worse than the 48.0% for the GA-trained constant goal networks. Thus the GA-trained networks do not need or benefit from explicit task goals.

Recall that in these implicit goal experiments the goal unit remained in the network but its activation was always zero. If we examine the ratio of the magnitude of the goal weights to all other weights again, we see that in the CRBP-trained implicit goal networks this ratio has dropped dramatically from 0.99 in the constant goal experiments to 0.11 here. Indeed, in the CRBP algorithm, weights coming from a completely inactive unit will never be changed and thus will remain at their initial random values. However, in the GA algorithm the weights to be mutated are simply selected randomly so the goal-related weights can still be modified. Thus, in the GA-trained implicit goal networks this ratio has actually risen from 0.70 in the constant goal experiments to 0.80 here. Once again we see that the GA is insensitive to the presence (or absence) of the goal.

The most telling evidence that the CRBP-trained networks benefit from explicit goals is that in two out of the five CRBP implicit goal runs, the learning actually produced hidden units that served a goal-like function. Figure 4.12 shows the activation of one such hidden unit (during the course of 300 actions) along with the sum of carbot's light readings. To achieve the seek goal the sum of the light readings must exceed 1.7, and to achieve the avoid goal the sum must fall below 0.6. Notice that the strongest peaks of activation for this hidden unit correlate with the achievement of the seek goal. In the constant goal experiments, the goal unit's activation in the input layer automatically switched sign to mark the achievement of a goal. In these implicit goal experiments, this external cue was removed, and in response, two of the CRBP-trained implicit goal networks developed their own internal cue for marking the achievement of seek goals.

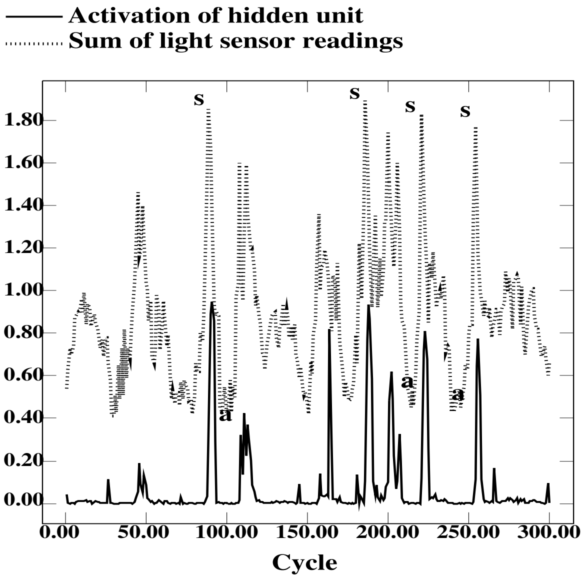

Figure 4.12: CRBP implicit goal: Development of a goal-like hidden unit when no explicit goal was present. An **s** indicates when a seek goal was accomplished and an **a** indicates when an avoid goal was accomplished.

4.3.5 Constant goals with delayed feedback

One small change in the CRBP learning procedure was made for this set of experiments. Because reward was obtained much less frequently (less than 1% of the time even after 500,000 cycles of training), the reward learning rate was increased from 0.3 to 1.0. Despite this change, when the immediate feedback about the light gradient was removed, the CRBP-trained networks were unable to converge on a successful strategy, receiving punishment on average 69.2% of the time. This performance was significantly worse than for the constant goal experiments where feedback was immediate ($p < 0.01$).

In contrast, the GA-trained delayed feedback networks were able to converge on several good solutions already discussed (such as the full-circle strategy). This was the second case where the GA's performance at 52.0% punishment was significantly better than CRBP's ($p < 0.05$). And once again, the GA's delayed performance did not significantly degrade from the constant goal case.

4.4 Discussion

Table 4.3 ranks the performance generalization scores for the experimental variations. For the one-step goals and the delayed feedback variations the performance levels for CRBP were significantly worse than for the GA. However in the other variations, even when the performance levels were similar, the two methods developed solutions quite different in character. Being a local method, CRBP is much more sensitive to the moment-to-moment changes in the environment and can thus use the constant goals to develop unique strategies tuned to each goal. In fact when no explicit goal is present, CRBP-trained networks will sometimes create their own goal-like units in the hidden layer. However this sensitivity to the environment can also be limiting. We saw that in each successive experimental variation, CRBP's performance significantly degraded to the point that it could not succeed with only delayed feedback about the light.

Rank	Method	Variation	Avg %pun	Strategy
1	CRBP	*constant*	41.2	semi-circles
2	GA	*one-step*	46.3	series of turns
3	GA	*constant*	48.0	full-circles
4	GA	*implicit*	48.7	two-point turns, stars
5	GA	*delay*	52.0	full-circles
6	CRBP	*implicit*	56.2	two-point turns
7	CRBP	*one-step*	62.6	none
8	CRBP	*delay*	69.2	none

Table 4.3: Overall ranking of generalization performance.

In comparison the GA, as a global method, tends to develop a single overall strategy that is applicable to both goals. More importantly, the GA's performance was quite robust across the experimental variations. But this insensitivity to environmental conditions can have a price. The GA's strategies were more likely to fall into ineffective cyclical behavior when carbot's position strayed from the vicinity of the light. However by lengthening the time spent on evaluating the fitness of individuals in the population this cyclic behavior could be lessened.

The respective strengths and weaknesses of these two adaptation methods are clearly complementary, suggesting that some hybrid of the two could be the most effective method. Because the GA globally samples the entire space of alternative solutions while CRBP locally searches the immediate neighborhood of a particular solution, the most straightforward form of hybrid would be to allow the GA to find a good starting point in the weight space and then use CRBP to do the fine tuning. Belew, McInerney, and Schraudolph did a number of experiments to test the feasibility of using a GA as a source of initial weights for gradient descent learning and found that this technique is quite effective (Belew et al., 1992).

Combining global and local adaptation methods such as the GA and CRBP is a promising avenue for further research into developing robotics controllers. But there is a caveat: the computational complexity of these hybrids can be extremely high. However, if such hybrid models can produce controllers that are both robust across gross environmental changes and yet sensitive to subtle features then the additional computational effort may be well worth it.

5

Towards planning

This chapter describes the final increment of the robotics control model. Recall that the primary hypothesis of this research is that the ability to plan develops from the more primitive capacity of reactive control. I will now offer a more concrete version of this hypothesis as it applies to the model investigated throughout this thesis.

Consider again the model's basic control architecture shown in Figure 5.1. As a side effect of learning how to react to the environment and the goals, this network may build up a record of its past states in the context layer. Note that there is no guarantee that this will be the case, but the capacity to do so is available in the recurrent connections. Thus immediately after a goal has been achieved, the contents of the hidden layer could conceivably reflect a generalized history of the environmental context encountered while achieving the goal. I will argue that the most successful controllers do indeed retain this sort of history and that this history can be the building block to emergent plans.

The specific hypothesis with respect to the model has now become: Given a robot controller based on the recurrent network shown in Figure 5.1, if it is provided with trigger-like goals, and trained with reinforcement learning, then upon achievement of a goal the hidden layer will contain information that could be used to plan for that goal. To prove this hypothesis I must show that these patterns of hidden activation can be used to guide behavior. First some changes will be made to carbot to better equip it for the task of planning.

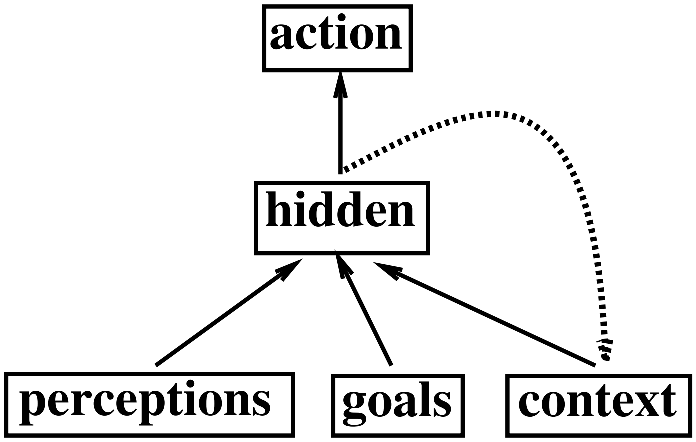

Figure 5.1: The robot controller architecture.

5.1 Enhancing carbot's sensors

Success in planning depends on the ability to anticipate future outcomes. Carbot's current sensors are quite primitive and provide little opportunity for easily anticipating the result of its actions. For instance, there are many positions in the playpen that produce roughly the same light readings. Some of these positions are close to a wall while others are not, and carbot's touch sensors are only triggered upon contact with a wall. Thus in these situations carbot will be unable to reliably avoid hitting the walls.

This has been called the *perceptual aliasing problem* and occurs when states in the world with important differences appear the same to the system (Whitehead and Ballard, 1991). There are two standard approaches to alleviating this problem. The first is to improve the sensors themselves. The second, as Kaelbling describes, is "to extend the system's inputs to include the last k percepts for some value of k" (Kaelbling, 1993). She further points out though, that as k gets large, the input space grows and dramatically increases the complexity of the overall control task. The recurrent memory of carbot's controller is a potentially more elegant solution to providing context for distinguishing between similar perceptual states because it is a fixed size and learning (rather than the designer) determines what will be retained in the memory. However, the information currently being supplied by the touch sensors is too sparse to effectively contribute to useful contextual memory development. To improve the potential for anticipation, the touch sensors will be replaced by sonar sensors.

Currently the sonar sensors have only been implemented in simulation, but I

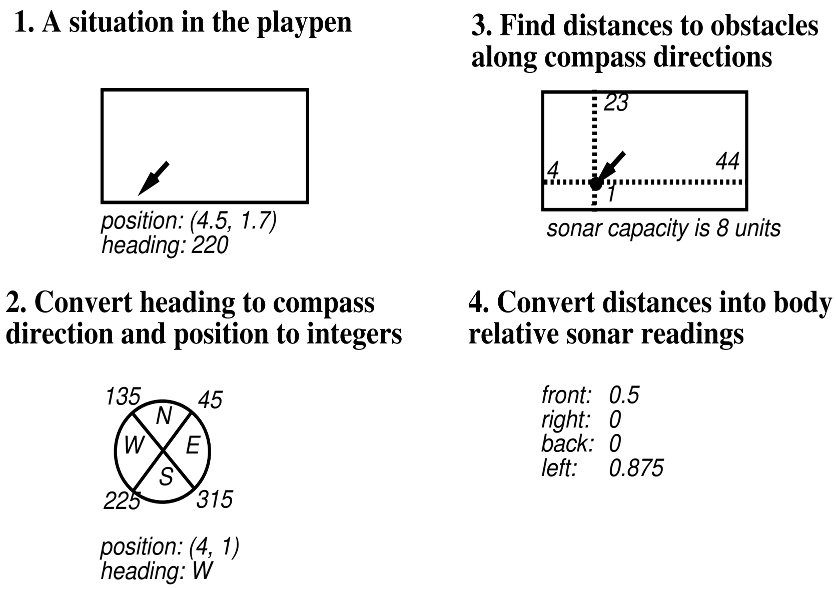

Figure 5.2: Method for obtaining simulated sonar readings.

eventually plan to equip the physical robot with them as well. The simulated carbot has four sonar sensors on its left, front, right, and back. Figure 5.2 summarizes the process used to obtain simulated sonar readings. First carbot's position and heading are determined. Second, the floor of the real-valued position is taken, converting it to integers, and the heading is converted to a compass direction. In the particular case shown, carbot is determined to be facing west because its heading is between 135 and 225 degrees. Third, the distance from the integer position to any obstacle along the compass directions is found. Finally the direction relative distances (north, south, east, west) are converted into body relative distances (left, front, right, back) and the sonar readings are calculated. If the distance to an obstacle is greater than the sonar range of 8 units, then the sonar reading will be 0. Otherwise the sonar reading is: $1 - (distance/range)$. In this way close obstacles will result in higher sonar readings than far obstacles. When the distance is 0, the sonar reading will be at its maximum value of 1 indicating contact with a wall. In the case shown, the west distance of 4 will become the front sonar reading of $1 - 4/8$, the north distance of 23 will become the right sonar reading of 0, and so on. With sonar capabilities carbot should be able to anticipate walls and more easily avoid hitting them.

5.2 Analyzing the controller's hidden representations

I have claimed that the hidden layer activations in the controller networks after a goal has been achieved will reflect the history of the states encountered during the task. Now it is time to analyze what these controller networks are actually encoding in their hidden layer. To do this we will use a technique called cluster analysis.

First, a controller network must be trained to a high level of performance. The network will have access to the light and sonar sensors, to periodic seek and avoid goals, and will use a hidden layer of size five. A network of this type was trained with CRBP for 200,000 actions, receiving punishment only 16.9% of the time by the end of training. This network was then tested for 1,000 actions and each time a goal was achieved the activations of the five units in the hidden layer were saved. This process resulted in 67 saved patterns. As an example, one pattern associated with seeking had the following values: (0.057256 0.341356 0.000200 0.525126 0.000034)

Cluster analysis constructed a hierarchy of partitions based on these 67 unique patterns of activation. This hierarchy was built by pairing a pattern with its closest neighbor in Euclidean space. This neighbor may be another pattern or an already constructed partition which is identified by the average profile of the patterns it contains. This process continued until every pattern was connected to a single hierarchical tree structure. Figure 5.3 shows the gross structure (into nine partitions) produced by the cluster analysis of the 67 saved patterns.

First let's consider the large trends depicted in this clustering. The network developed unique strategies for seeking and avoiding the light. Typically for avoiding the light, the network used one-point turns created by a series of BL turns followed by a series of FR turns. For seeking the light, the network used combinations of BR and FL turns. The clustering reflects this difference in strategies—most of the avoid patterns are grouped in the top half of the tree (in clusters 1, 2, 4 and 5) while most of the seek patterns are grouped in the bottom half of the tree (in clusters 6, 7, 8, and 9). Figures 5.4 and 5.5 depict representative behavior from each of the nine clusters.

The exceptions to the separation of avoid and seek patterns are illustrative. Cluster 3 is one of these exceptions; it consists of five seek patterns, but has been clustered in the avoid half of the tree. These five seek patterns were created when the network used, in succession, a BL and then a FR move at the end of the task to reach the light. Neither of these actions is typically chosen during a seek task, and except for these five seek instances these two actions only occur in succession during avoid tasks. The resulting patterns of activation were thus closer to avoid patterns created from

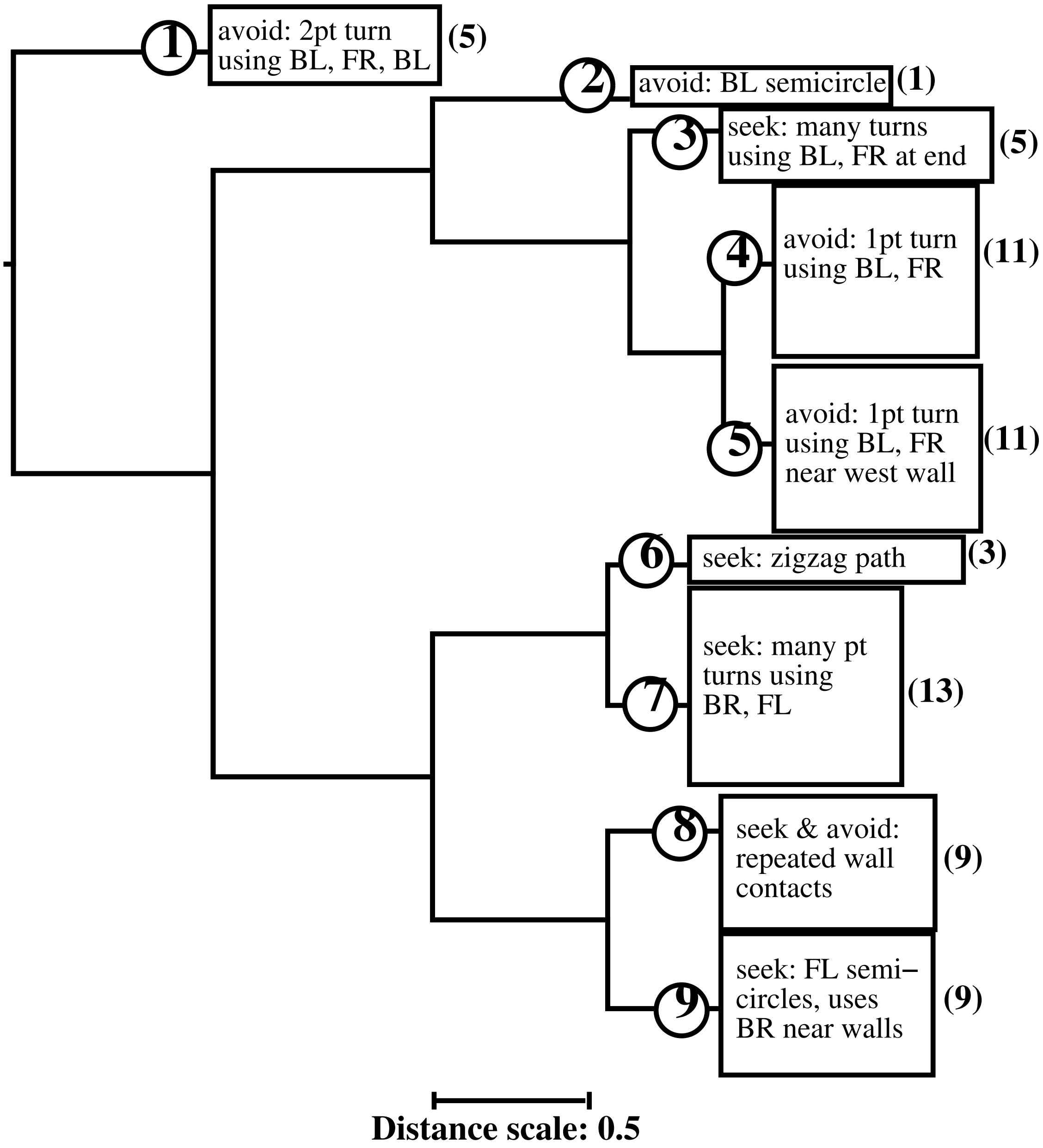

Figure 5.3: Annotated cluster diagram of the hidden layer activations after goal achievement in a controller with sonar capabilities. Numbers in parentheses indicate the size of the cluster. The substructure of each cluster is not shown.

this same series of actions.

Cluster 8 is the other exception; it contains of mixture of seek and avoid patterns and is grouped in the seek half of the tree. The avoid patterns of this cluster were all created after carbot was stuck in the lower-left corner. During this time the controller tried using FL and BR actions to extricate carbot and since these actions are typically associated with seeking the light the patterns were closer to other seek cases. The behavior that led to the seek patterns of this cluster were also related to problems with navigating in corners.

Both of these exceptions provide evidence that these hidden activations do reflect some history of the states preceding goal accomplishment. If the clustering was based solely on the the final perceptual state then avoid behaviors and seek behaviors would not be clustered together since their final perceptual states are quite different. Successful seek behaviors will place carbot near the lower-left corner facing the light while successful avoid behaviors may place carbot almost anywhere in the environment as long as it is facing away from the light.

Now let's look more closely at the relationships between the avoid clusters. Clusters 4 and 5 contain the majority of the avoid cases and there is only a subtle difference between the two. In cluster 4, the representative behaviors have a longer BL phase than FR phase to the one-point turn, and in cluster 5, the opposite is the case. For example, the hidden pattern associated with the sequence of actions BL-BL-BL-FR is attached to cluster 4 while the hidden pattern associated with the actions BL-BL-FR-FR-FR is attached to cluster 5. The behaviors in cluster 1 are much like those in cluster 4 except that the FR phase brought carbot too close to the north wall and an additional BL action was needed. Cluster 2 is a bit of an anomaly containing only one instance of a BL semi-circle.

The representative behaviors from the seek clusters exhibit even more variety. In cluster 6, the final approach to the light was achieved by alternating between FL and FR actions creating a zigzag path. In cluster 7, frequent switching of direction created star-like patterns, while in cluster 9 smoother one-point turns were used.

In the previous chapter we saw that, as a group, the control networks exhibited a wide range of behaviors (full circles, semi-circles, one-point turns, two-point turns, star patterns etc), but each particular network tended to exhibit a single behavioral style. With the addition of sonar capabilities, a single controller can develop a much larger repertoire of behavior. Furthermore, the cluster analysis reveals that the hidden activations at the time of goal achievement reflect these behavioral nuances. Because these hidden representations encode, to some degree, the structure of the behavioral strategies produced by the control networks they hold promise as a first step towards planning. For this reason I will refer to them as *protoplans*. The next section describes

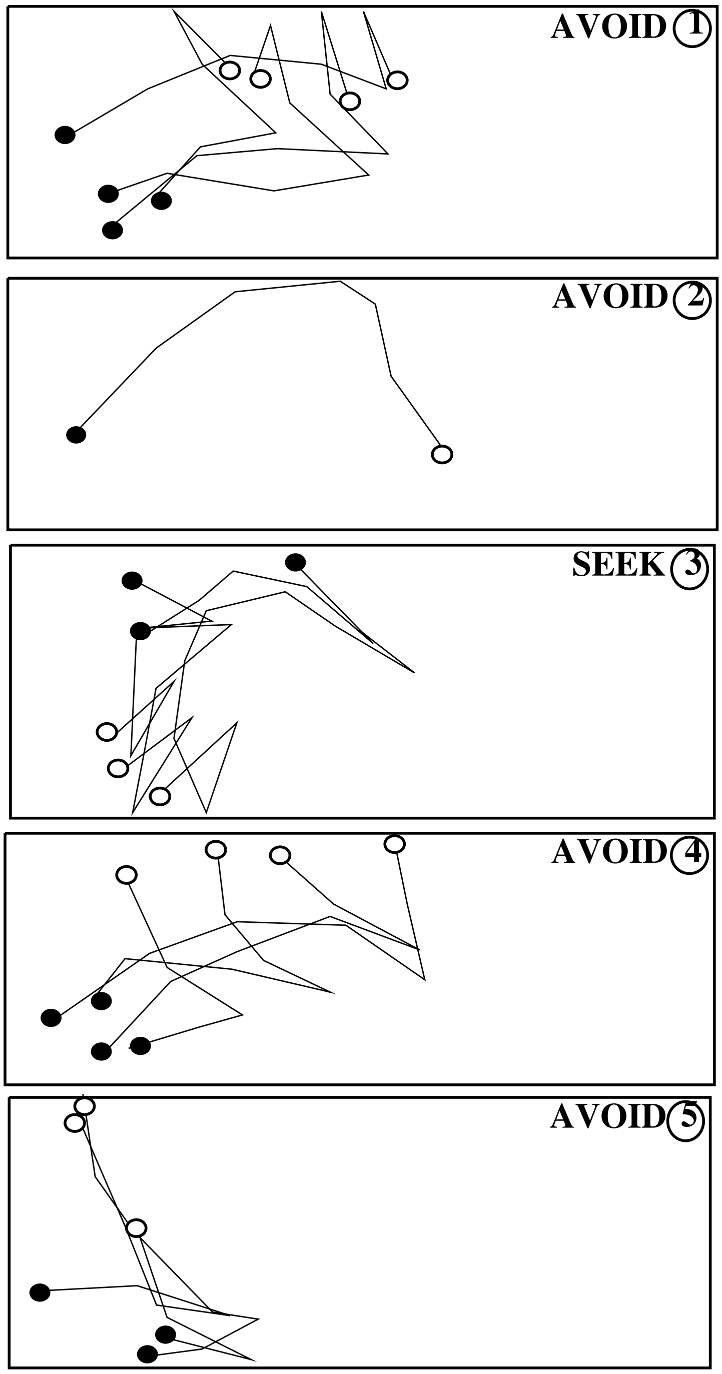

Figure 5.4: Representative behavior from clusters primarily associated with avoiding the light which is located in the lower-left corner. Closed circles indicate where the behavior began and open circles indicate where the goal was achieved.

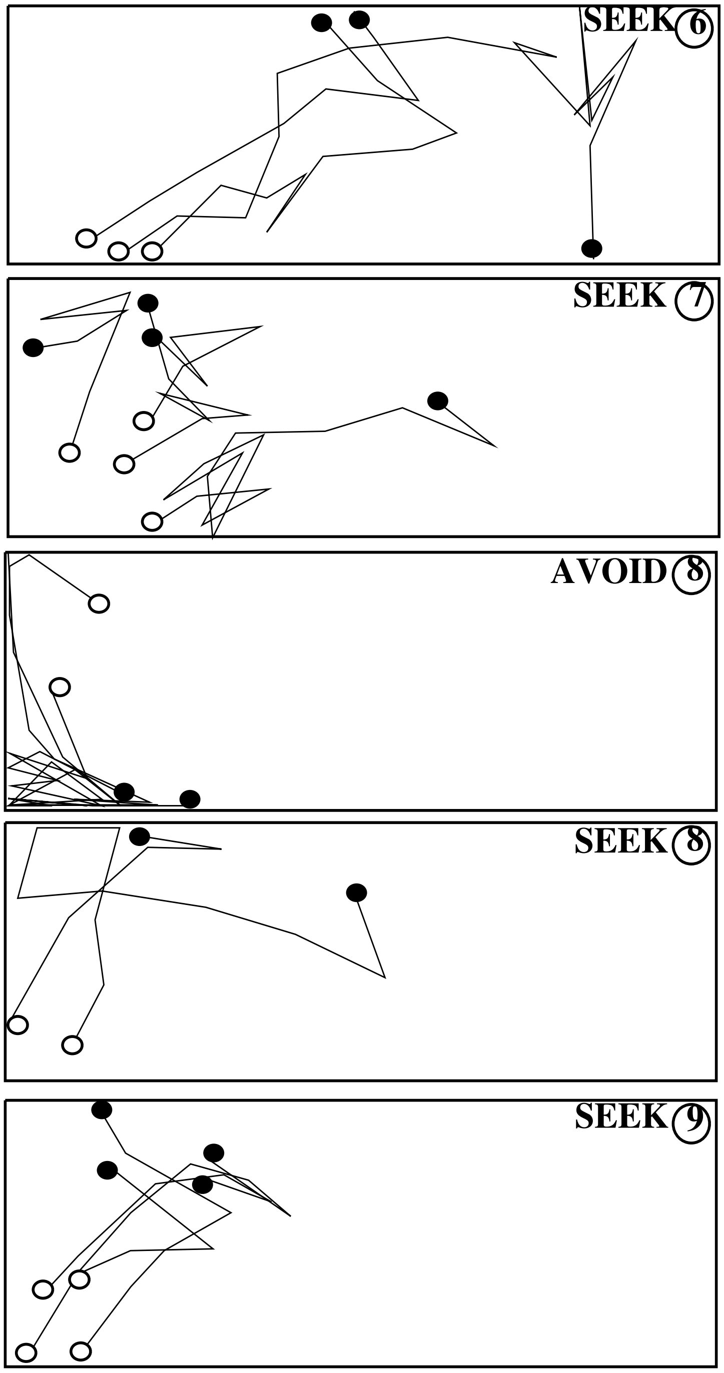

Figure 5.5: Representative behavior from clusters primarily associated with seeking the light which is located in the lower-left corner. Closed circles indicate where the behavior began and open circles indicate where the goal was achieved.

a method used to examine whether these protoplans can be put to practical use to guide action decisions.

5.3 Method for applying protoplans

To investigate the efficacy of protoplans, a transfer-of-learning task will be explored: can the protoplans learned in one controller be transferred to a second controller to provide guidance as it learns from scratch on the same task? To answer this question, a control network will be trained with goals to produce a set of protoplans. As a means to transfer the protoplans to the new network, a memory will be created that associates the input states that immediately preceded the achievement of a goal with their protoplan. Finally a new control network will begin learning without goals, but with access to this protoplan memory. Using its own input state, this new control network can retrieve an appropriate protoplan out of the memory based on the original network. A careful examination of the performance of this second control network, which was given protoplans, will elucidate what role, if any, protoplans play in its behavior. Figure 5.6 summarizes this experimental method. A more detailed explanation follows.

For step 1 in Figure 5.6 a carbot controller with sonar sensors, light sensors, and periodic goals was trained for 300,000 actions using CRBP. The hidden layer once again contained five units. Next for step 2 of Figure 5.6, the controller was tested for 1,000 actions. During this testing phase, the hidden layer activations at the time of goal achievement—the protoplans—were saved. In addition, the five input states that preceded the achievement of the goal were saved. An input state consists of the current sonar readings, light readings, and goal value. Table 5.1 shows a trace of a consecutive set of input states during the test phase.

At time step 30 of this trace a seek goal was initiated and at time step 35 it was accomplished. Thus at time step 35 the activations in the hidden layer are considered to be a protoplan and are saved. We will refer to this particular protoplan as pp_{35}. The input states from time steps 31 through 35 are also saved since these states immediately preceded the goal achievement. Because these precursor states include the sensor readings and the goal values we will refer to them as $s + g_{31}$, $s + g_{32}$, ..., $s + g_{35}$. After 1,000 time steps had been processed in this manner, 134 protoplans and 654 input states were obtained (the average number of actions needed to accomplish a goal was actually less than 5 at 4.88). This concludes step 2 of Figure 5.6.

Next, the memory was constructed that provided the means of transferring the expertise obtained during the training of the original control network to a new control

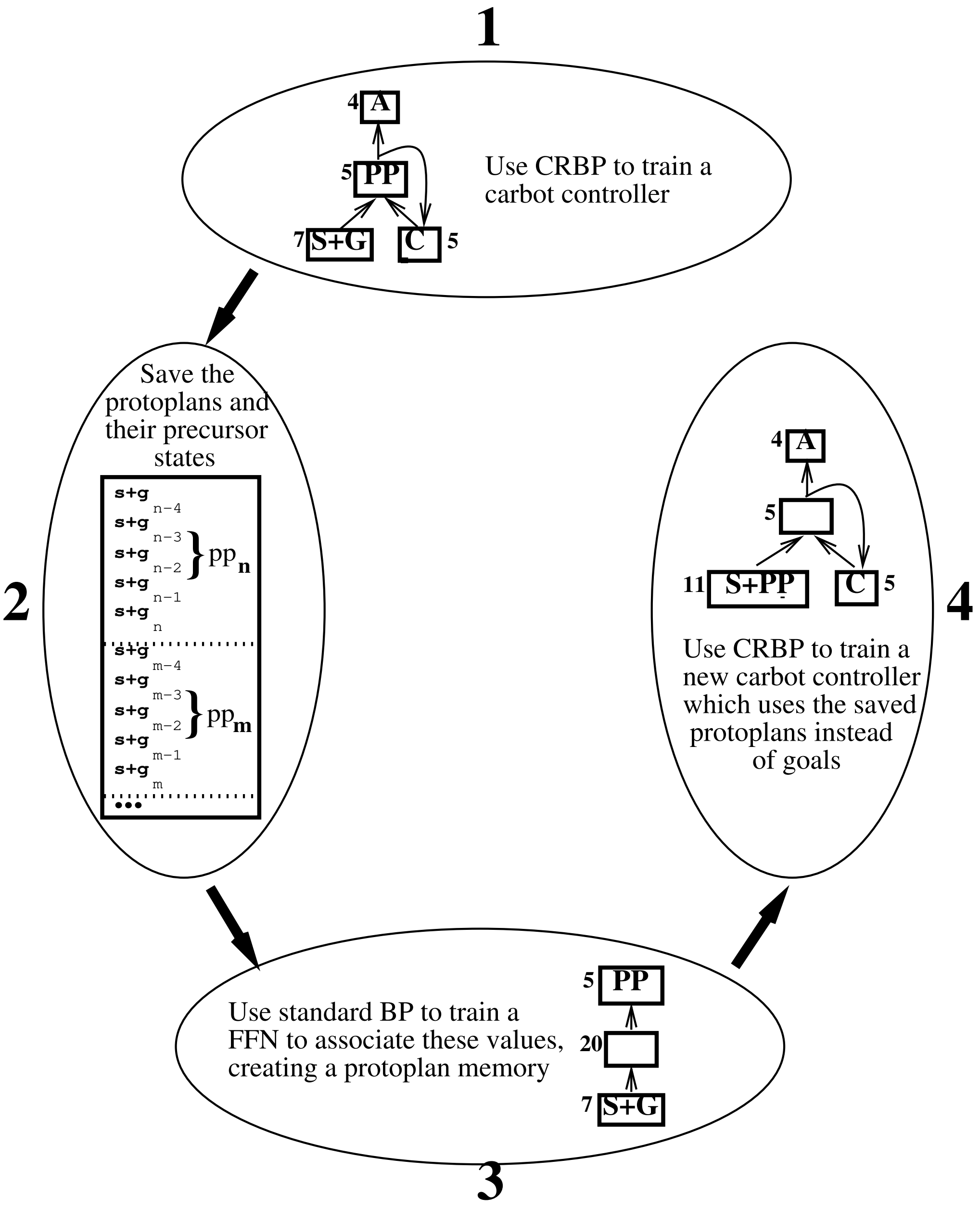

Figure 5.6: Experimental method for investigating protoplan guided behavior. Key to the figure: S=sensor readings, G=goal, PP=protoplan, C=context, and A=action. See text for details. The numbers beside each network architecture indicate the size of the layers.

	Action	Sonar				Light		
Time	taken	Left	Front	Right	Back	Left	Right	Goal
29	BL				.20	.29	.22	-1
30	FL				.14	.23	.26	+1
31	FL		.17			.43	.25	+1
32	BR			.17		.63	.27	+1
33	FL			.14		.72	.27	+1
34	FL		.14			.93	.52	+1
35	FL	.14	.20			.93	.91	+1
36	BL	.12	.12			.87	.60	-1

Table 5.1: Trace of a controller's input states after training. Empty locations indicate zero values.

network. This protoplan memory was implemented as a feedforward neural network. The input layer contained seven units (one for each sensor reading and one for the goal), the hidden layer had 20 units, and the output layer had five units (one for each value of the protoplan). When given a precursor state as input the protoplan memory was trained with standard back-propagation to return the subsequent protoplan as output. Returning to the trace of Table 5.1, this means that for pp_{35} all of the following associations were trained:

$s + g_{31} \rightarrow pp_{35}$
$s + g_{32} \rightarrow pp_{35}$
$s + g_{33} \rightarrow pp_{35}$
$s + g_{34} \rightarrow pp_{35}$
$s + g_{35} \rightarrow pp_{35}$

The protoplan memory network was trained with a learning rate of 0.05 for 80 epochs. An *epoch* constitutes one pass through the entire training set. Since the training set consisted of 654 pairs, this equated to 52,320 training steps. After training, the average sum-squared error for each input-output pair was 0.17. Note that this particular association task cannot be performed perfectly because the same input state is often associated with numerous protoplans. To obtain a reasonable level of performance this memory network learned to produce generalized versions of the protoplans as output. This concludes step 3 of Figure 5.6.

The final step of the method to investigate protoplan applicability was to train a new controller with access to the protoplan memory. We have seen in Chapters 3 and 4 that goals can have a powerful influence on the final behavior of a control network. To better pinpoint the role protoplans might play in guiding behavior, goals were omitted from this second controller's input. However, this creates a problem: to retrieve protoplans from the memory we need both the current sensor readings and the current goal. The simulator maintains an internal variable that tracks the current goal and this was used to complete the input state needed for retrieving protoplans from the memory. The steps of this training process are enumerated below:

1. Query the sensors
2. Use the sensor readings plus the simulator's internal record of the current goal to query the protoplan memory
3. Place the sensor readings and the retrieved protoplan on the input layer of the control network
4. Forward propagate these values and continue with the CRBP algorithm

The control network with protoplans was also trained for 300,000 actions to facilitate comparisons between it and the original control network. This concludes the explanation of the transfer-of-learning method. In the next section the experiments based on this method will be described.

5.4 Accessibility of protoplan information

Previously, the cluster analysis showed that the protoplans encoded information that closely correlated with variations in the controller's behavior. But how accessible is this information in the protoplan? Can newly developing networks use the protoplans to improve performance? There are basically three possible answers to these questions. First, the information in the protoplan may be completely inaccessible. In this case we would expect the performance of controllers with protoplans to be no better than controllers trained with only sensors as input. The second possibility is that just the goal information in the protoplan is accessible. In this case we would expect the performance of controllers with protoplans to be equivalent to controllers with goals. The final possibility is that both the goal and further information may be accessible in the protoplan. In this case we would expect the performance of controllers with protoplans to be better than controllers trained with goals.

To determine which of these scenarios is correct, three variations on the input given to control networks were tested: sensors only, sensors and goals, and sensors and protoplans. For each variation five trials were done starting from random weights and training with CRBP for 300,000 cycles. From the five networks that were trained with sensors and goals, the most successful network was chosen as the source of the protoplans. During testing this controller was punished only 11% of the time.

	Avg %punishment			
Variation	Training	SD	Testing	SD
S	58.1	0.4	57.0	7.1
S+G	29.3	2.8	12.9	5.3
S+PP	26.3	4.4	13.9	5.9

Table 5.2: Punishment performance in protoplan experiments. The variations differ with respect to what was available as input (S = sensors, G = goals, and PP = protoplans). The training value reflects the average punishment received over all 300,000 cycles of learning. The testing value reflects the average punishment received during 1,000 actions after training.

Table 5.2 summarizes the performance of the three variations. Both controllers trained with goals and controllers trained with protoplans were significantly better than controllers trained with only sensors ($p < 0.01$). From this result it is clear that the first scenario, where the protoplans provide no information, is incorrect because controllers with protoplans out-performed controllers with only sensory information. To resolve the issue of the accessibility of protoplan information we need to consider the differences between networks trained with goals and networks trained with protoplans. The results show that there were no significant differences in the final performance of these two variations. Thus, it appears that the second scenario, where only goal information is accessible in the protoplans, is correct. Notice, though, that the overall punishment received during training is slightly less for controllers with protoplans. Although both variations are ultimately equivalent in performance, perhaps the controllers with protoplans reach a high level of performance earlier in the learning.

To investigate this hypothesis, the average amount of punishment was re-calculated every 1,000 actions of training. By plotting these values a profile of the course of learning was obtained. Figure 5.7 shows the speed of convergence for a network with protoplans versus a network with goals. In this particular case the controller with protoplans converged more quickly than the controller with goals. To quantify this difference across the networks, a convergence point was designated at 35% punishment. For each network, the cycle of training where this point was reached and never exceeded again was noted. Table 5.3 shows the average number of actions needed to reach this level of performance for controllers with goals and controllers with protoplans. The protoplan controllers were consistently faster to converge.

However this difference in learning speeds was not significant due to one protoplan

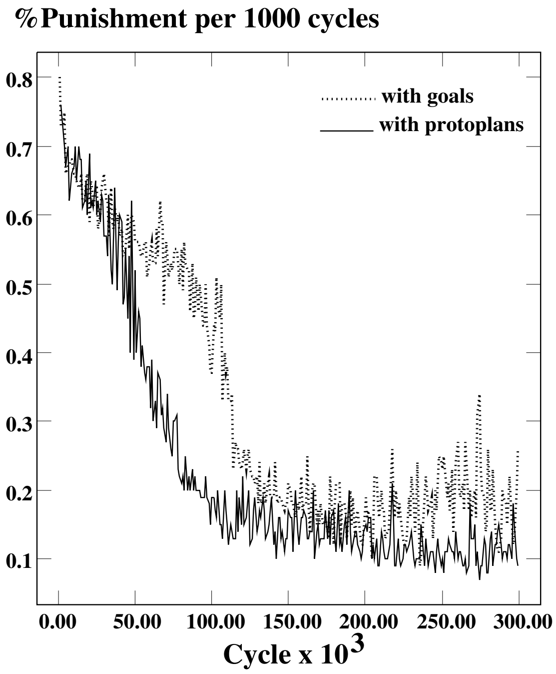

Figure 5.7: Comparison of learning speeds in a controller with goals versus a controller with protoplans.

Variation	Avg training cycle when pun < 35%	SD
S+G	114,600	24,745
S+PP	80,800	27,087

Table 5.3: Speed of convergence in protoplan experiments.

network which was quite slow to converge relative to the others. If this outlier is removed from the set, then the average time to convergence for protoplan networks drops to 69,000 and the standard deviation drops to 7,070. If the comparison is redone with the outlier removed, then the difference in learning speeds is significant ($p < 0.01$). The fact that networks with protoplans and without direct access to goals can learn the task more quickly than networks with goals suggests that the third scenario is correct—there is more than goal information accessible in the protoplans. The next section will look closely at the contents of the protoplans.

5.5 How do protoplans guide behavior?

Table 5.4 summarizes the primary strategies used by the goal-trained network that was the originator of the protoplans and the protoplan-trained networks based upon it. Of the five goal-trained networks, only GnetA developed a semi-circle based strategy; the other solutions relied on alternating turns. Of the five protoplan-trained networks, three developed similar semi-circle seek strategies, and two of these also developed similar semi-circle avoid strategies to the original GnetA. Therefore, semi-circle strategies developed more frequently in the networks using protoplans created by a semi-circle based network than the networks relying only on goal guidance. This offers some evidence that the protoplans may have had a global influence on the development process. To understand the local effects of protoplans, this section will dissect the operation of controller PPnet1. This particular network was chosen because it was one of the two protoplan-trained controllers that developed strategies most like GnetA.

	Strategies	
Controller	Seek	Avoid
GnetA	FL semi-circle	BL semi-circle
PPnet1	FL semi-circle	BL semi-circle
PPnet2	FL semi-circle	BL semi-circle
PPnet3	BR,F turns	BL,FR one-point turn
PPnet4	FL semi-circle	BL,FR one-point turn
PPnet5	FR semi-circle	BR semi-circle

Table 5.4: Strategies learned by the originator of the protoplans (GnetA) and the controllers which used them (PPnet1 - PPnet5).

On each cycle of processing, PPnet1 queries the protoplan memory with its current sensor readings, allowing the protoplan input to change at every time step. Figure 5.8 depicts the contents of PPnet1's protoplans over the course of 55 time steps. When PPnet1's behavior is examined over these same time steps it is clear that units 2 and 3 of the protoplan are strongly associated with seeking the light, while units 1 and 4 are strongly associated with avoiding. Unit 5 was uniformly inactive so has been omitted.

During seeking-the-light phases, unit 2 immediately reaches a nearly maximum level of activation and remains there, acting like a goal unit. Unit 3 initially rises and in some instances abruptly fluctuates between 0.6 and 0.4 before falling to a low level again (one such example has been circled). During avoiding the light phases, units 1 and 4 tend to rise and fall in-step with one another, though unit 4 is always more

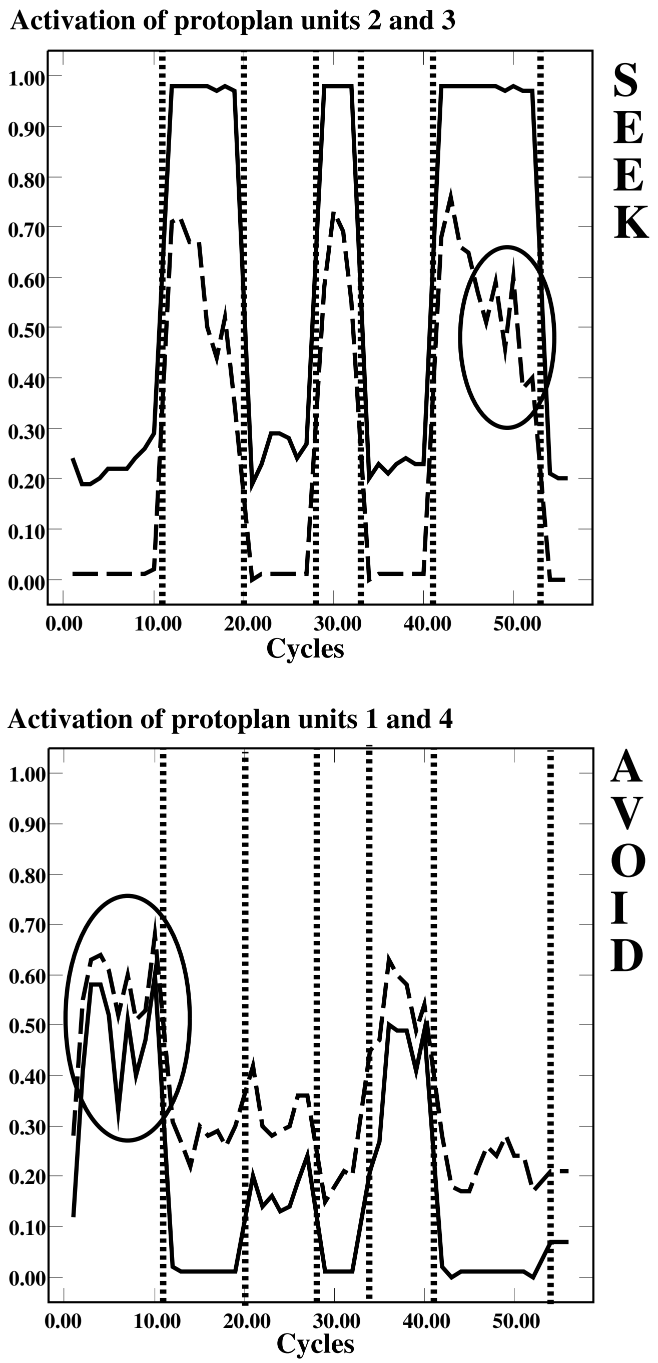

Figure 5.8: Activation pattern of protoplan units. Of the five units in the protoplan layer, only units two (solid curve) and three (dashed curve) are highly active during seek behavior. Only units one (solid curve) and four (dashed curve) are highly active during avoid behavior. The vertical lines indicate when a goal was achieved. The circled portions will be examined in detail. Note that each figure of this sort which follows reflects the same time period.

active. Units 1 and 4 also experience instances of quick fluctuations in the activation values (one such example has been circled). We will concentrate on the behavior of PPnet1 during the time steps associated with these circled fluctuation points.

Consider first the fluctuations of unit 3 during the seek behavior that begins at time step 42 and ends at time step 53. Recall that PPnet1 uses FL semi-circles to navigate to the light. However, if it begins approaching the light from the far-right end of the playpen, a succession of FL actions will force carbot into the south wall long before reaching the light. In these situations, the controller has learned to instead alternate between FL and FR actions, zigzagging carbot towards the light while sidestepping the south wall. It is precisely during this zigzag motion that unit 3's fluctuations occur. A drop in unit 3's activation correlates with a FR action and a rise correlates with a FL action.

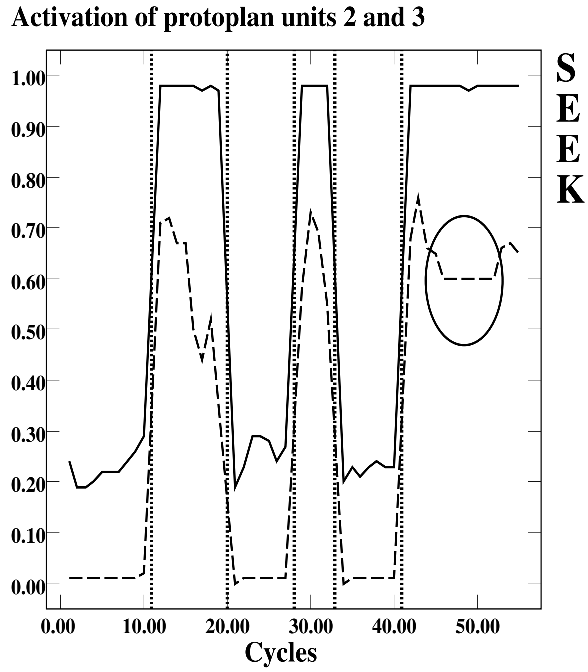

Figure 5.9: Activation pattern of protoplan units associated with seeking the light after being disturbed.

This correlation is suggestive, but does not provide clear-cut evidence that the protoplan is being used to guide the behavior. To reveal whether this is the case, we will disturb the protoplan memory during the zigzag phase (time steps 47-51) and note the effect on behavior. If the controller responds to a drop in unit 3's activation by switching to a FR action, then by fixing unit 3's activation and removing the fluctuations we may be able to eliminate all the FR actions during these time steps.

Figure 5.9 shows how the circled portion of the seek protoplan was adjusted; every other aspect of the environment remained exactly as it had been before.

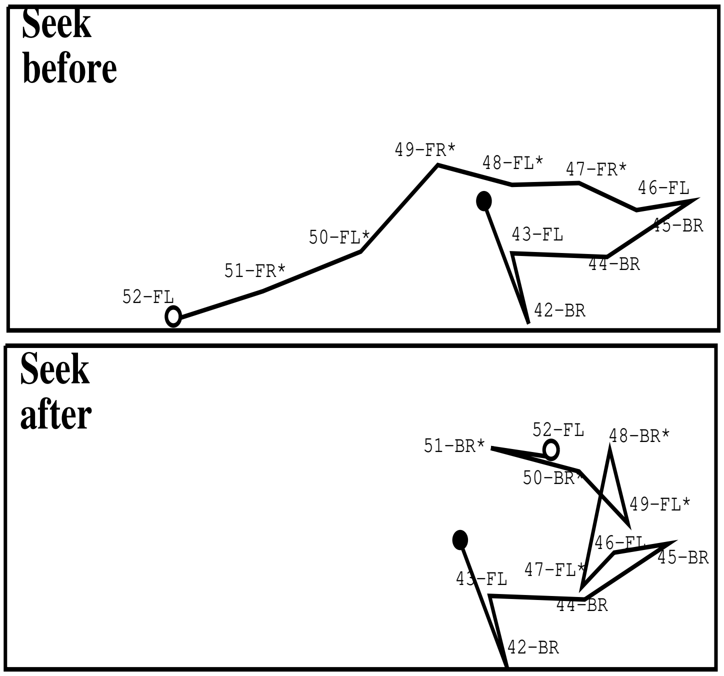

Figure 5.10: Seek behavior in network with protoplans before and after disturbance. Asterisks indicate the actions that correspond with the circled portions of the traces of the protoplan contents.

Figure 5.10 depicts the behavior that occurred before and after this disturbance of the seek protoplan. The disturbance began at time step 47. Originally the controller produced a FR action at this point, but after the disturbance it produced a FL action. Then because the new FL action brought carbot near the south wall, the controller responded at step 48 with a BR action. Throughout the disturbance to the protoplan, FL and BR actions were used and the controller did not produce a FR action. Similar fluctuations in later time steps were also examined in this manner, and in each case, fixing unit 3's activation eliminated the zigzag behavior. This is strong evidence that the protoplan is the cause of zigzag behavior.

Next, let's consider whether disturbances to the avoid units of the protoplan will produce equally dramatic results. PPnet1 uses BL semi-circles to avoid the light. If carbot nears a wall while backing away from the light, a FR action is used and then the BL semi-circle is resumed. The fluctuations in units 1 and 4 correlate with these switches of direction from BL to FR. When the activations fall, a FR action is produced, and when the activations rise, a BL action is produced.

PPnet1 proved to be much less sensitive to disturbances to units 1 and 4 than it was to units 2 and 3. Raising both units' level of activation to 0.7 did not eliminate the FR actions. Lowering both units' level of activation to 0.5 did not affect the BL actions either. Only by lowering the activation of both units to 0.1 was any effect in behavior produced. Figure 5.11 shows how the circled portion of the avoid protoplan was adjusted. Because this disturbance occurred early on in time, the subsequent trace of the avoid protoplan is quite different from the original version.

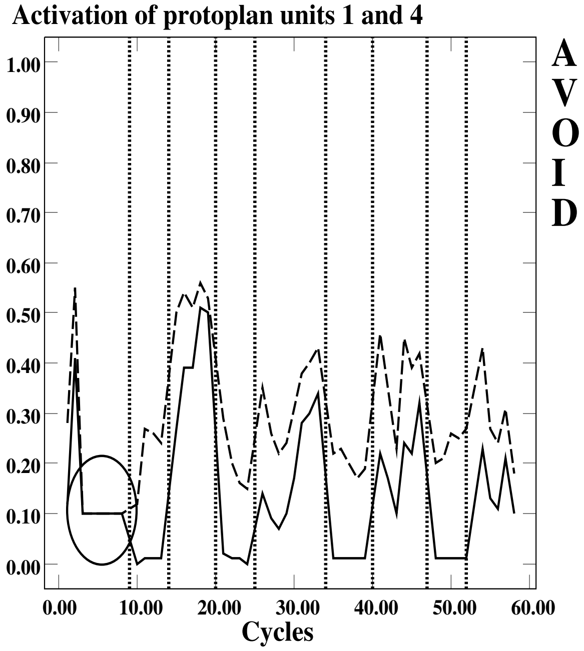

Figure 5.11: Activation pattern of protoplan units associated with avoiding the light after being disturbed.

Figure 5.12 depicts the behavior that occurred before and after this disturbance to the avoid protoplan. By drastically lowering the level of activation in both avoid protoplan units, all of the BL actions were eliminated. This response is impressive despite the fairly severe disturbance required to elicit it. An adjustment to the protoplan caused a complete reversal of the standard avoid behavior—a series of BLs was replaced by a series of FRs. As with the seek protoplan, invoking the same disturbances in the avoid protoplan at different points in time affected the corresponding behavior in the same way.

Each of the other protoplan networks were tested for sensitivity to disturbances in their protoplans. Table 5.5 shows that the protoplan controllers that developed

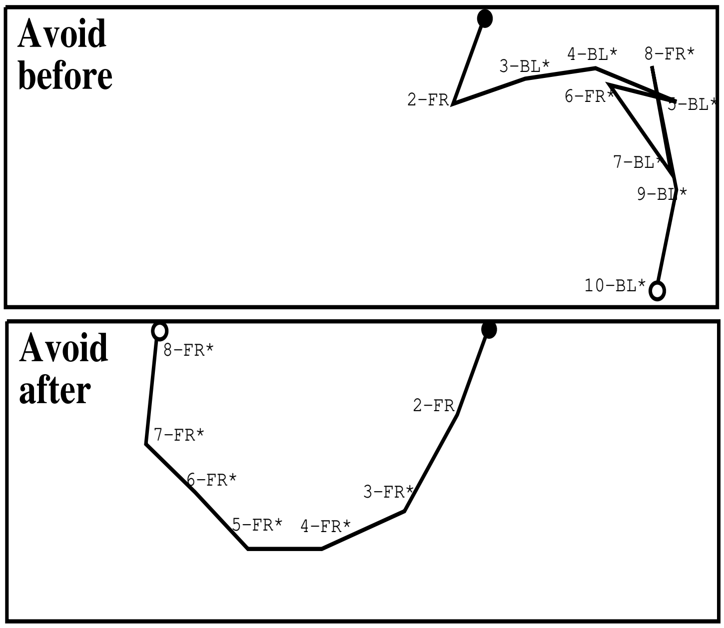

Figure 5.12: Avoid behavior in network with protoplans before and after disturbance. Asterisks indicate the actions that correspond with the circled portions of the traces of the protoplan contents.

strategies most similar to the original protoplan generator were also the most susceptible to disturbances. Because PPnet3 and PPnet5 used such different strategies from the original network, the fluctuations in the protoplans were not as closely correlated with changes in their behavior. Still in certain limited situations, the protoplans seemed able to provide some guidance and disturbances affected the behavior.

5.6 Discussion

In summary, the standard action patterns of both the seek and avoid strategies were consistently disrupted when disturbances were created in the normal fluctuations of the protoplans. Furthermore, the seek behavior was more susceptible to disruption than the avoid behavior. One explanation for the stability of the avoid behavior is that the switch from the BL semi-circle to a FR action is an essentially reactive maneuver. The sonar sensors provide ample warning of an impending collision so the controller need not rely on the protoplan for guidance about obstacles. Here the avoid protoplan primarily provides confirmation for the reaction. Still when the avoid

Controller	Sensitivity	
	Seek	Avoid
PPnet1	***	*
PPnet2	***	*
PPnet3	**	
PPnet4	***	
PPnet5	**	

Table 5.5: Sensitivity of seek and avoid strategies to disturbances in the protoplan activations. The number of asterisks depicts the degree of sensitivity. PPnet1 and PPnet2 developed behaviors very much like the network that originated the protoplans. PPnet4 developed a similar seek strategy, but a different avoid strategy from the original network. PPnet3 and PPnet5 developed their own unique strategies for both tasks.

protoplan's fluctuations were amplified enough, the obstacle reaction was induced without the proper stimulus. In contrast, the zigzag aspect of the seek strategy does not have as direct a basis in sensor readings. Following the light gradient requires that the two light readings be compared, while responding to an obstacle depends on a single sonar value. The seek protoplan provides the proper timing for executing the zigzag pattern and when this timing was disrupted the controller fell back on the basic semi-circle strategy. Because the zigzag behavior is more abstract than obstacle avoidance, the controller relies more on the protoplan to produce it.

The transfer-of-learning experiments were quite successful. Protoplans developed in one network were applied in other networks and speeded the learning process. However the success of this enterprise depends on the structure of the original goal-directed network's hidden activations. When these hidden patterns are distinct (as in Figure 5.13) they should serve as a good foundation for protoplans, but when they are closely clumped (as in Figure 5.14) they will be uninformative and ineffective.

These experiments offer a glimpse of a new style of plan that is context-dependent, dynamic, and offers indirect guidance about action decisions. The protoplans we examined emerged from a network that was involved in a close interaction with its environment. Important moments in this interaction were marked by systematic fluctuations in the protoplan values. When a new controller using these protoplans developed a similar interaction with the environment, these fluctuations provided guidance and confirmation about the proper timing of changes in behavior. The context-dependent character of the protoplans is reflected in the fact that the protoplans were most useful to controllers that developed the same strategy as the original controller. The kind of timing information the protoplans provided was indirect—no

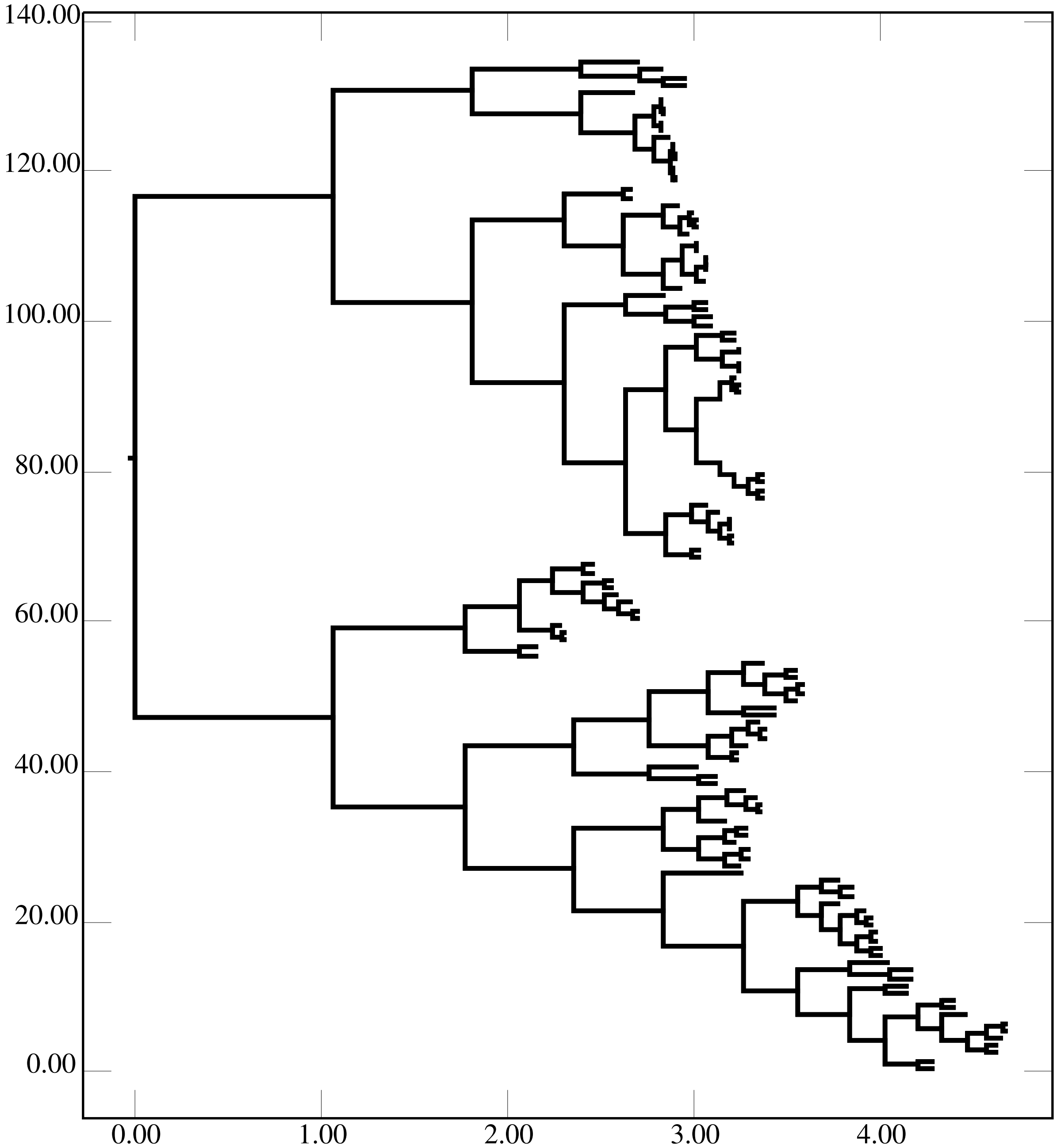

Figure 5.13: A highly distinct set of hidden layer activations. The network used to produce this cluster provided the protoplans for the previous set of experiments.

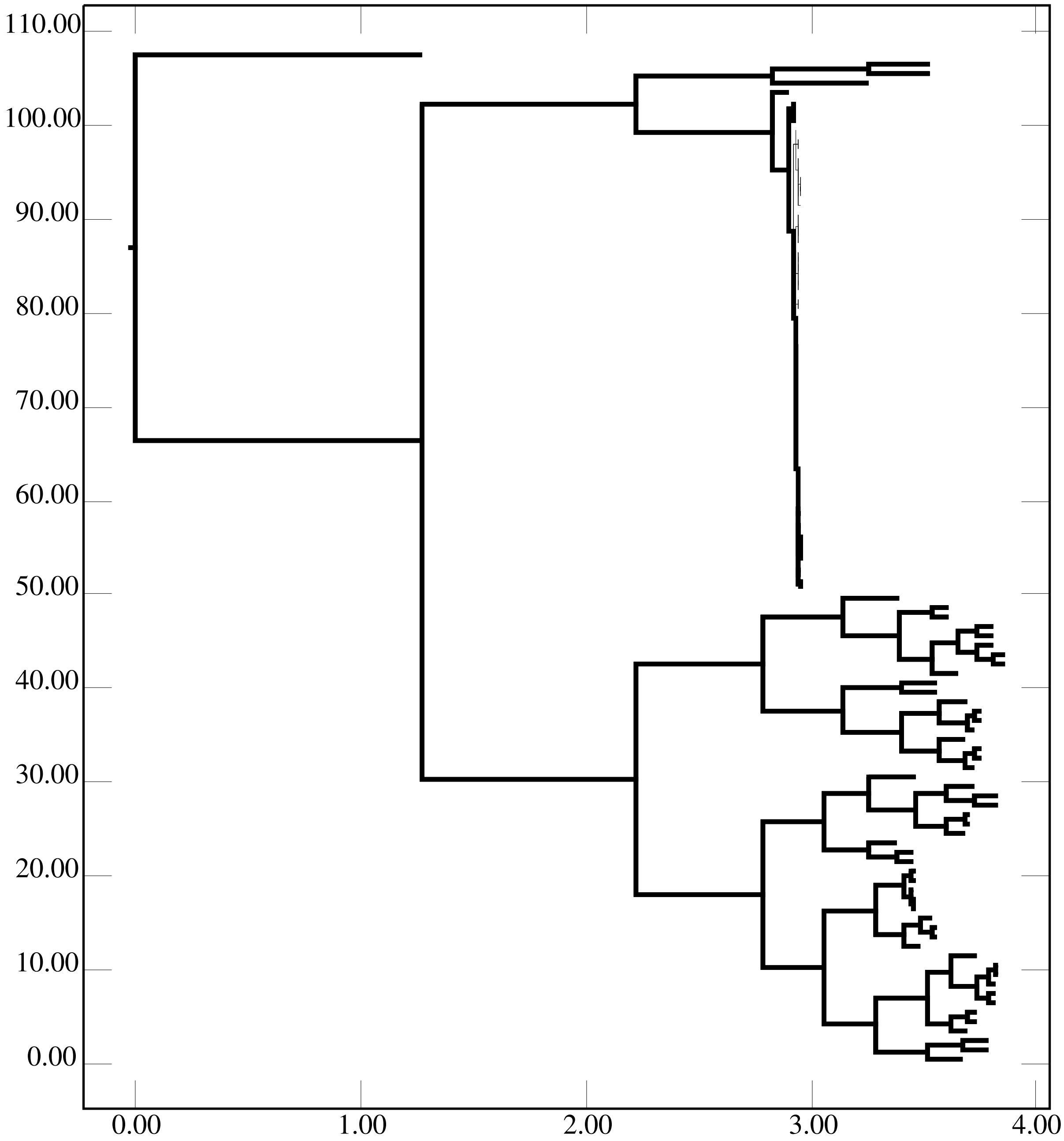

Figure 5.14: An indistinct set of hidden layer activations. All of the activation vectors associated with successful seeking behavior are closely clumped.

specific actions were indicated—but knowing when to change one's behavior may be more important than knowing how to change it.

6

Conclusion

The model investigated in this thesis was developed in an incremental fashion on two planes—the capacities of the robot and the level of external guidance provided. The initial robot resided in a one-dimensional world and was equipped with a single light sensor and two touch sensors. Next the model was extended to an actual robot, called carbot, that was equipped with two light sensors and four touch sensors. Finally, in simulation, carbot's touch sensors were upgraded to sonar sensors.

In step with these improvements to the robot, the level of external guidance provided to the model was manipulated. Initially, the effect of goals was examined. Next, two contrasting methods of providing evaluations to the model, CRBP and a GA, were compared. Finally, the development and use of a new form of plan-based guidance was considered. Throughout these incremental investigations, the emphasis has been on adaptation from limited designer bias.

In Chapter 3, I suggested a new form of goal which I termed a trigger. Rather than using goals to explicitly describe the desired outcome, these triggers provided an abstract cue about the current phase of the task, either seeking or avoiding the light. Varying the persistence of these trigger goals from continuous, to intermittent, to implicit significantly changed every aspect of the controller networks—the performance, the pattern of weights, the use of the context memory, and the course of learning. These goal experiments clearly demonstrated that the connectionist approach to control allows the task demands to be the primary force in shaping behavior, both the external behavior of the robot and the internal behavior of the controller network. The trigger style goal proved to be an effective means of providing external guidance without pre-specifying a solution to the task. However, even this abstract type of cue can be overpowering when the domain is quite simple. The goal information must be properly blended with perceptions to produce robust behavior.

Just as trigger-like goals provide an appropriately abstract kind of guidance, reinforcement learning offers indirect and sketchy evaluations of actions in the form of reward and punishment. In CRBP, the gradient descent method, the evaluation of an action affects the learning rate and the chosen target for back-propagation. When rewarded, CRBP strongly pushes the network to produce the same action again in similar situations, and when punished, it less strongly pushes the network to produce the opposite action in similar situations. In the genetic algorithm, a whole series of actions are tested to produce a single global evaluation called the fitness. Individual controllers that receive high fitness marks are more likely to have mutated copies of their network structure added to the population.

Chapter 4 contrasted the strengths and weaknesses of these two reinforcement learning methods. Despite similar levels of performance across a number of experimental variations, the character of the solutions produced by the two methods was quite unique. Being a local method, CRBP was much more sensitive to the moment-to-moment changes in the environment. When a goal was provided, CRBP-trained controllers developed two different, though related, strategies for seeking and avoiding the light (such as a forward-left semi-circle for seeking and a backward-left semi-circle for avoiding). When the goal was removed, some CRBP-trained controllers created their own triggering unit in the hidden layer. Not surprisingly for a local method, CRBP failed to succeed when the evaluation of the actions was delayed until the time of goal achievement. In comparison, as a global method, the GA was relatively insensitive to the dynamics of the environment and thus its performance was more robust across the variations. Regardless of whether a goal was present, the GA developed a single overall strategy that was applicable to both the seek and avoid the light tasks (such as a forward-left full-circle).

The previous increments of the robot control model showed that a controller which is given trigger-style goals and is trained with CRBP will develop behavior that is unique to each goal. As a side effect of learning how to produce this dynamic behavior, the contents of the short-term memory should reflect a generalized history of the environmental context encountered while achieving the goals. If this is indeed the case, then the hidden activations at the time of goal accomplishment can serve as the building blocks to plans. This is specifically how the ability to plan could develop from the more primitive capacity of reactive control.

Armed with these insights gained from the earlier increments, the final extension of the model, towards planning, was undertaken in Chapter 5. Sonar capabilities were added to the robot to ensure that the hidden representations would contain an adequate amount of information for planning. By replacing the proximal information given by the touch sensors with the distal information given by the sonar sensors, the controller was better able to anticipate the future. This enrichment in perception led to an enhancement of action. With sonar capabilities, a single controller was able to develop a much more varied repertoire of behavior. The cluster analysis revealed that the structure of the hidden patterns reflected these nuances and because of this the patterns could serve as protoplans.

To prove that protoplans could be used to guide behavior, a learning transfer experiment was conducted. The protoplans constructed in one network were stored in an associative memory and retrieved by a new network as it learned the task from scratch. In this way the strategies discovered in one controller biased the strategies developed in a new controller. Another agent, rather than a human designer, was able to direct the learning process.

Controllers with protoplans but no goals were able to converge more quickly to successful solutions than controllers with goals. This showed that the structural subtleties seen in the cluster analysis of the protoplans were accessible to some degree. Changes in the protoplans over time revealed that particular fluctuations in the protoplan values were highly correlated with switches in behavior. In some instances, very minor disturbances to the protoplan at these fluctuation points severely disrupted the normal pattern of behavior. Thus protoplans were playing a key role in determining the behavior.

The success of these protoplan experiments supports the new set of intuitions about planning encompassed by the reactive planning view. Rather than static, standalone procedures, plans can be seen as dynamic, context-dependent guides, and the process of planning may be more like informed improvisation than deliberation. It is not fruitful to spend processing time reasoning about an inherently unpredictable world. With the protoplan model, a new protoplan can be recomputed *on every time step*. Although each protoplan offers only sketchy guidance, any more information might actually be misleading. Once the chosen action is executed, the subsequent perceptions are used to retrieve a new, more appropriate protoplan. Therefore it is possible to continually replan based on the best information available—the system's current perceptual state. Furthermore, plan use in the protoplan model is continuous. The controller learns to strongly depend on some protoplans and to be loosely guided by other protoplans. Planning is not an all or nothing process here. Protoplans can play a variable role in all action decisions.

The transfer experiments were a fairly direct means of determining how using protoplans as input could affect behavior. But because protoplans are so dependent on the context in which they were created, ultimately I believe that they should remain a part of the system which constructed them. To this end, the controller should be able to save and generalize over its own protoplans. Figure 6.1 shows a proposed architecture for the on-line storage and retrieval of protoplans. This architecture was inspired by work on Hebbian Recurrent Networks (Dennis and Wiles, 1993). In this architecture, a protoplan memory can be updated on every time step to reflect the system's ever changing summary of the current situation. In addition, a generalized version of this summary can be used to affect the current action decision.

Within this type of system, protoplans could serve as the communication link between various modules. By being grounded in the environment, protoplans can provide information about the dynamics of the world to higher order modules not directly connected to perception. Furthermore, these higher order modules could create higher order protoplans leading to more complex levels of control.

As I continue to explore further increments of this model by increasing the complexity of the robot and the tasks, neither of the adaptation methods on their own will be adequate. As the task difficulty and network size increases, CRBP's effectiveness will diminish. In contrast, the GA will remain effective, but its insensitivity to the fine environmental details make it a poor source of protoplans. In Chapter 4 we saw that the respective strengths and weaknesses of CRBP and GAs were complementary, and in future work I will use a hybrid of the two methods. The GA will be used to globally sample the space of alternative controllers, and CRBP will be used to locally search the immediate neighborhood of a particular solution. Such a hybrid should produce controllers that are both robust across gross environmental changes and yet sensitive to subtleties as well.

This thesis has described and demonstrated a possible origin of plans. Protoplans emerged from the situated activity of an agent through the formation of a generalized history of the states encountered while achieving a goal. This generalized history in the short-term memory arose as a natural side effect of having to make reactive decisions. This first step towards planning developed directly from the more primitive capacity of reactive control.

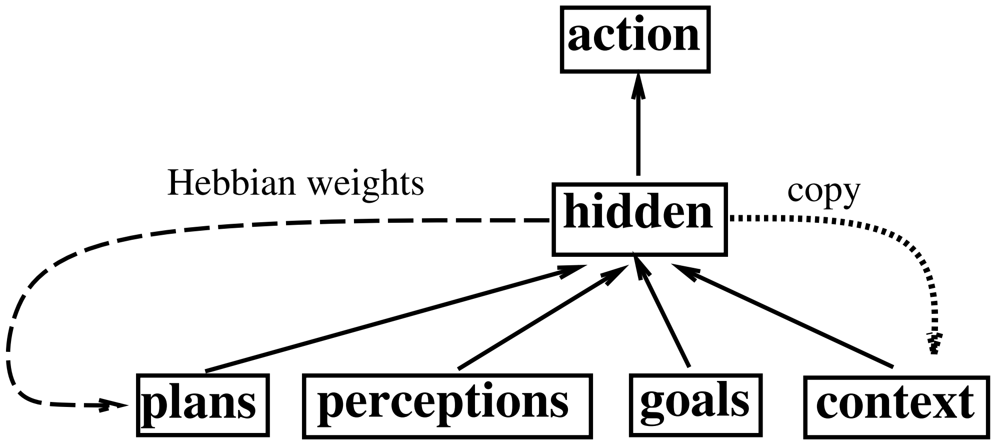

Figure 6.1: Architecture for storing protoplans for use within a single system.

References

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

Agre, P. E. (1988). *The Dynamic Structure of Everyday Life*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science.

Agre, P. E. (1993). The symbolic worldview: Reply to Vera and Simon. *Cognitive Science*, 17(1):61–70.

Barto, A. G. (1989). Connectionist learning for control: An overview. Technical Report 89-89, University of Massachusetts, Department of Computer and Information Science.

Beer, R. D. (1990). *Intelligence as Adaptive Behavior: An Experiment in Computational Neuroethology*. Academic Press, Inc., San Diego, CA.

Beer, R. D. and Gallagher, J. C. (1992). Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior*, 1(1):91–122.

Belew, R. K., McInerney, J., and Schraudolph, N. N. (1992). Evolving networks: Using genetic algorithms with connectionist learning. In Langton, C. G., Taylor, C., Farmer, J. D., and Rasmussen, S., editors, *Artificial Life II*, pages 511–547, Redwood City, CA. Addison-Welsley.

Blank, D. S., Meeden, L. A., and Mashall, J. B. (1992). Exploring the symbolic/subsymbolic continuum: A case study of RAAM. In Dinsmore, J., editor, *The Symbolic and Connectionist Paradigms: Closing the gap*, pages 113–148. Lawrence Erlbaum Associates, Hillsdale, NJ.

Braitenberg, V. (1984). *Vehicles: Experiments in Synthetic Psychology*. MIT Press, Cambridge, MA.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23.

Brooks, R. A. (1991). Intelligence without representation. *Artificial Intelligence*, 47(1):139–159.

Chalmers, D. J. (1990). The evolution of learning: An experiment in genetic connectionism. In *Proceedings of the 1990 Connectionist Summer School*, Palo Alto, CA. Morgan Kaufmann.

Chapman, D. (1987). Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377.

Chapman, D. (1991). *Vision, Instuction, and Action.* MIT Press, Cambridge, MA.

Clark, A. (1993). *Associative Engines: Conectionism, Concepts, and Representational Change.* MIT Press, Cambridge, MA.

Cliff, D., Harvey, I., and Husbands, P. (1993). Explorations in evolutionary robotics. *Adaptive Behavior*, 2(1):73–110.

Dennis, S. and Wiles, J. (1993). Integrating learning into models of human memory: The hebbian recurrent memory. In *Proceedings of the Fifteenth Annual Meeting of the Cognitive Science Society*, pages 394–399, Hillsdale, NJ. Lawrence Erlbaum Associates.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14:179–212.

Firby, R. J. (1989). *Adaptive Execution in Complex Dynamic Worlds.* PhD thesis, Yale University, Department of Computer Science. Technical Report YALEU/CSD/RR 672.

Fitzpatrick, M. J. and Grefenstette, J. J. (1988). Genetic algorithms in noisy environments. *Machine Learning*, 3:101–120.

Gasser, M. (1993). The structure grounding problem. In *Proceedings of the Fifteenth Annual Meeting of the Cognitive Science Society*, pages 149–152, Hillsdale, NJ. Lawrence Erlbaum Associates.

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

Griffin, D. R. (1984). *Animal Thinking.* Harvard University Press, Cambridge, MA.

Harnad, S. (1990). The symbol grounding problem. *Pysica D*, 42:335–346.

Harp, S. A., Samad, T., and Guha, A. (1989). Towards the genetic synthesis of neural networks. In Schaffer, J. D., editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 360–369, Palo Alto, CA. Morgan Kaufmann Publishers, Inc.

Harvey, I. (1993a). Evolutionary robotics and SAGA: The case for hill crawling and tournament selection. In Langton, C. G., editor, *Artificial Life III*, Redwood City, CA. Addison-Welsley.

Harvey, I. (1993b). Issues in evolutionary robotics. In Meyer, J.-A., Roitblat, H., and Wilson, S., editors, *Proceedings of the second international conference on the simulation of adaptive behavior*, Cambridge, MA. MIT Press.

Husbands, P., Harvey, I., and Cliff, D. (1993). Analysing recurrent dynamical networks evolved for robot control. In *Proceedings of the Third IEE International Conference on Neural Networks*. IEE Press.

Jordan, M. I. (1989). Serial order: A parallel, distributed processing approach. In Elman, J. L. and Rumelhart, D. E., editors, *Advances in Connectionist Theory: Speech*. Lawrence Erlbaum Associates, Hillsdale, NJ.

Kaelbling, L. P. (1993). *Learning in Embedded Systems*. MIT Press, Cambridge, MA.

Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64.

Laird, J. E. and Rosenbloom, P. S. (1993). Integrating execution, planning, and learning in Soar for external environments. In Rosenbloom, P. S., Laird, J. E., and Newell, A., editors, *The Soar Papers: Research on Integrated Intelligence*, volume 2, pages 1036–1043. MIT Press, Cambridge, MA.

Laird, J. E., Yager, E. S., Tuck, C. M., and Hucka, M. (1989). Learning in tele-autonomous systems using Soar. In Rodriquez, G. and Seraji, H., editors, *Proceedings of the NASA Conference on Space Telerobotics*, volume 3, pages 415–424, Pasedena, CA. NASA Jet Propulsion Laboratory, California Institute of Technology.

Martin, F. (1992). Mini board 2.0 technical reference. MIT Media Lab, Cambridge MA 02139.

Meeden, L. A., McGraw, G., and Blank, D. (1993). Emergence of control and planning in an autonomous vehicle. In *Proceedings of the Fifteenth Annual Meeting of the Cognitive Science Society*, pages 735–740, Hillsdale, NJ. Lawrence Erlbaum Associates.

Mozer, M. C. (1989). A focused beck-propagation algorithm for temporal pattern recognition. *Complex Systems*, 3:349–381.

Newell, A. (1980). Physical symbol systems. *Cognitive Science*, 4:135–183.

Newell, A. and Simon, H. A. (1972). Computer science as empirical inquiry: Symbols and search. *Communications of the ACM*, 19:113–126.

Norman, D. A. (1993a). Approaches to the study of intelligence. In Rosenbloom, P. S., Laird, J. E., and Newell, A., editors, *The Soar Papers: Research on Integrated Intelligence*, volume 2, pages 793–812. MIT Press, Cambridge, MA.

Norman, D. A. (1993b). Cognition in the head and in the world: An introdution to the special issue on situated action. *Cognitive Science*, 17(1):1–6.

Pearlmutter, B. A. (1989). Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1:263–269.

Pfeifer, R. and Verschure, P. (1992a). Beyond rationalism: Symbols, patterns and behavior. *Connection Science*, 4(3–4):313–325.

Pfeifer, R. and Verschure, P. F. (1992b). Designing efficiently navigating non-goal-directed robots. In Meyer, J., Roitblat, H., and S., W., editors, *From Animals to Animats*.

Ram, A. and Santamaria, J. C. (1993). A multistrategy case-based and reinforcement learning approach to self-improving reactive control systems for autonomous robtic navigation. In *Proceedings of the Second International Workshop on Multistrategy learning*.

Rich, E. (1983). *Artificial Intelligence*. McGraw-Hill, New York, NY.

Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning internal representations by error propagation. In McClelland, J. and Rumelhart, D., editors, *Parallel Distributed Processing*, volume I, pages 318–362. MIT Press, Cambridge, MA.

Schnepf, U. (1991). Robot ethology: A proposal for the research into intelligent autonomous systems. In Meyer, J.-A. and Wilson, S. W., editors, *From Animals to Animats*, pages 465–474, Cambridge, MA. MIT Press.

Smolensky, P. (1988). On the proper treatment of connectionism. *Behavioral and Brain Sciences*, 11:1–74.

Suchman, L. A. (1987). *Plans and Situated Action*. Cambridge University Press, Cambridge.

Verschure, P. F., Krose, B. K., and Pfeifer, R. (1992). Distributed adaptive control: the self-organization of structured behavior. *Robotics and Autonomous Systems*, 9:181–196.

Waltz, D. L. (1991). Eight principles for building an intelligent robot. In Meyer, J.-A. and Wilson, S. W., editors, *From Animals to Animats*, pages 462–464, Cambridge, MA. MIT Press.

Whitehead, S. D. and Ballard, D. H. (1991). Learning to perceive and act. *Machine Learning*, 7(1):45–83.

Whitley, C., Dominic, S., Das, R., and Anderson, C. W. (1993). Genetic reinforcement learning for neurocontrol problems. *Machine Learning*, 13:259–284.

Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280.

Wilson, S. W. (1991). The animat path to AI. In Meyer, J.-A. and Wilson, S. W., editors, *From Animals to Animats*, pages 15–21, Cambridge, MA. MIT Press.

Yamauchi, B. M. (1993). Dynamical neural networks for mobile robot control. NRL Memorandum Report AIC-033-93, Naval Research Laboratory, Washington, D.C.

Yamauchi, B. M. and Beer, R. D. (1994a). Integrating reactive, sequential and learning behavior using dynamical neural networks. Submitted to the *Third International Conference on Simulation of Adaptive Behavior*.

Yamauchi, B. M. and Beer, R. D. (1994b). Sequential behavior and learning in evolved neural networks. *Adaptive Behavior*, 2(3):219–246.