Abstraction in Control Learning

Richard Yee

Department of Computer and Information Science
University of Massachusetts, Amherst, MA 10003

COINS Technical Report 92-16
March 1992

Abstract

Efficient learning requires strong generalization biases, which are often provided to learning agents through carefully designed input representations. Human novices, however, are capable of using experience in a task to develop their own expert representations. This report is a dissertation proposal examining how agents can acquire learning biases through the construction of abstract representations. Efficient learning and planning call for representing specific experiences in terms of abstract features and concepts that reflect the goals and dynamics of tasks. Recent AI research has begun to incorporate established results from control theory, which studies the interaction between agents and dynamical systems or environments. Learning methods related to *dynamic programming* (DP) address the problem of finding useful sequences of control actions, without detailed instruction from a teacher. This report proposes guidelines for using *function approximation* to improve the scaling properties of DP-based learning, and it proposes two approaches for forming abstractions of tasks. The first approach uses abstract *state values* to induce abstract classes of task states. The second approach seeks to learn such state classes by constructing hierarchical connectionist networks whose units act as abstract features or concepts. Both approaches are designed to facilitate control over memory resources, allowing learning to accelerate from early rote memorization to more globally-scaled generalization.

This material is based upon work supported by the Air Force Office of Scientific Research, Bolling AFB, under Grant AFOSR-89-0526 and by the National Science Foundation under Grant ECS-8912623.

Contents

1 Introduction

How does a novice become an expert in a task? How does one master what was once difficult or impossible? And how does learning bootstrap new knowledge and skills out of existing abilities? The research described in this proposal explores such questions by focusing on the representations used by agents in learning to perform tasks. In particular, I consider the problem of enabling artificial agents to construct their own *abstract representations* of tasks. Such representations simplify both learning and performance by structuring information: crucial large-scale concepts and features of a task become distinguishable from smaller-scale details.

Abstractions are necessary because time and memory are limited resources. In performing a complex task, rarely can the full ramifications of potential actions be apprehended before it becomes necessary to act. Decisions must be made with respect to the most significant aspects of the overall task. Consequently, an agent can benefit greatly from representations that facilitate decision-making under time and memory constraints. Abstractions support efficient performance by highlighting aspects of situations that are of primary concern for achieving the agent's goals.

Abstractions are also necessary for efficient learning. Normally, future situations will never perfectly match ones faced in the past. A learner must be able to extract essential qualities from specific experiences, thereby generalizing to other situations that are in some sense "similar." Even when identical experiences are repeated, it is usually infeasible to represent each one as an isolated event. In most cases, therefore, a failure to generalize adequately implies a failure to learn.

It is typically possible to generalize in diverse ways from any finite amount of experience, even if extensive. A learning agent thus requires constraining *biases* to ensure that its generalizations accord with the goals and dynamics of its task. Learned abstractions should help provide such constraints because they will have been tailored by experience to fit the given task. Abstractions provide a vocabulary of features and concepts that should accelerate the process of translating specific experiences, episodes, into a body of semantic knowledge covering a broad range of task situations, including ones never previously encountered.

Suitably designed abstractions can greatly improve the efficiency of both learning and performance in a task. Unfortunately, creating such representations by hand remains, in most cases, a black art. The approaches in this proposal seek to automatically construct abstract representations that provide useful simplifications of tasks. The abstractions consists of Boolean features and concepts formed through the experiences of a learning agent engaged in a task. A principal consequence of the learned abstractions should be the acquisition of high levels of performance at accelerated rates, beyond what would be expected from agents relying solely on the representations typically provided by human designers.

1.1 Learning and optimal control

The proposed research will be conducted within the context of learning to perform *optimal control tasks*. Studying learning within an established framework for control is a departure from the paradigm of *learning from examples* which has dominated research in artificial intelligence (AI) in the areas of both *machine learning* and *connectionism* (*artificial neural networks*). Barto (1991; 1990) discusses the integration of learning and control, pointing out issues that, until recently, have received relatively little attention. Learning in optimal control directly addresses the problem of designing agents that can learn to interact with dynamic systems and environments. Such an approach also provides new perspectives on the research areas of both *supervised* and *unsupervised* learning from examples.

Supervised tasks, where training examples are input-output pairs, view learning as *function approximation*, *pattern recognition*, or *concept acquisition*. Unsupervised tasks, where training examples are inputs alone, view learning as *data clustering*, *data analysis*, or *feature/concept formation*. Although recent research has made significant contributions in these areas, a concentrated focus on classifying bodies of data might cause one to lose sight of important practical considerations. In classification, one necessarily makes many implicit and explicit assumptions about the quality and presentation of data and about the goals and strategies of learners. If research becomes removed from real contexts in which learning must operate, then concern arises that unrealistic assumptions could limit the usefulness any results achieved. There is a risk of searching only "under lampposts."

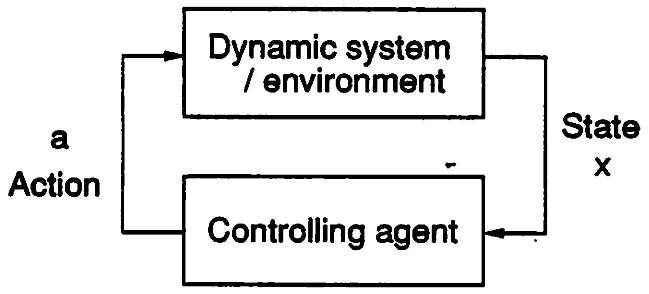

Figure 1: The basic feedback control architecture. The agent seeks to guide the system's behavior along desirable state trajectories by executing sequences of actions.

Control may be a case in point. Control tasks present both complexities and opportunities for learning that have not been commonly addressed. In particular, this proposal examines function approximation with an emphasis on issues faced in an optimal control framework adopted for the research. Although forming abstractions is consistent with a common theme in learning, namely, transforming data representations to improve subsequent processing (Dietterich, 1982; Hinton, 1989), it appears that most learning research on function approximation methods fails to address the full range of issues present in optimal control. Learning from examples is undeniably important, but if agents are to learn autonomously to act in dynamic environments, then optimal control issues need to be faced directly.

Optimal control Control tasks are characterized by the interaction between an *agent* (the controller) and a *dynamic system* or environment (the plant). Figure 1 is a block diagram of the basic feedback control architecture. The agent receives information about the *state* of the system and produces *control actions* that influence the *state transitions* of the system. The agent's control objective is to produce action sequences that will guide the system's behavior along desired *state trajectories*. In optimal control, desired trajectories extremize some performance measure relative to all other trajectories. Optimal trajectories are typically unknown in advance of actually controlling the system. Hence, not only must the agent learn how to employ actions to direct the system, in optimal control it must also discover the best trajectories for the system to follow.

Although learning in this context can be very demanding, the problem is ubiquitous. Agents should be able to learn autonomously, i.e., when there is no *teacher* to provide feedback that is both timely and comprehensible to the learner. For example, the game of chess is easily defined and has specific goals. However, no oracle exists to point out the proper lines of play in every situation. In a non-optimizing framework, a learner might try to reproduce the demonstrated abilities of an expert. In general, however, one does not wish to restrict learning only to cases in which an external entity can, not only solve the task, but also provide the right level of instruction (cf. Jordan & Rumelhart, 1990). It is often difficult to obtain good sources of guidance for solving tasks which are nonetheless easy to formulate. A variety of problems in areas ranging from motor control, robot navigation, and speech production to game-playing, planning, and heuristic search appear to be natural candidates for modeling as optimal control tasks.

Exploiting prior knowledge Unlike unsupervised tasks, learning in optimal control requires agents to find good solutions to *specific* problems, and, unlike supervised tasks, agents must learn even if persistent, high-quality feedback is unavailable. Given such stringent conditions, one may expect learning in optimal control to be slow in general. Hence, straightforward *weak* or *tabula rasa* approaches are unlikely to be practical for many tasks of realistic complexity. Several lines of research, including the one in this proposal, are directed toward dealing with such scaling issues.

One important approach is to use *strong* methods that enable agents to efficiently acquire and exploit available expertise regarding a particular task. Such methods can provide a foundation that greatly improves the efficiency of learning in optimal control. For example, a learner might be given a *model* or *domain theory* describing the controlled system. Models for simulating systems enable look-ahead search or planning, processes that can be faster, more comprehensive, and less risky than analogous exploration using real systems. Perhaps more significantly, models and domain theories provide knowledge about the structure and function of a system that can sometimes be used in a reverse or retrospective manner to analyze or explain events through techniques such as *explanation-based learning* (EBL) (Mitchell et al., 1986; DeJong & Mooney, 1986; Laird et al., 1986; Minton et al., 1989), *goal regression* (Waldinger, 1976), or gradient-estimation (Werbos, 1990; Jordan

& Jacobs, 1990; Jordan & Rumelhart, 1990; Bachrach, 1991; Gullapalli, 1991). Such analytical techniques can be efficient, resulting in high levels of performance on the basis of relatively little training information.

In addition to knowledge about the system, another type of potentially useful information consists of control or evaluation heuristics or other suboptimal control or evaluation abilities, e.g., *nominal* controllers or value functions that put the learner "in the right ballpark" for finding optimal strategies. Finally, a learner might be efficiently led to discover complex control strategies by solving sequences of progressively more difficult tasks in a manner akin to the technique of *shaping*, which is used in animal learning. In general, the use of task-specific information can be expected to greatly accelerate learning processes, although a heavy dependence on such methods might preclude the achievement of truly optimal performance. In many cases, such a tradeoff would be more than acceptable as learning might be otherwise impractical.

Nevertheless, explicitly providing agents with extensive knowledge of their tasks will not be a major focus of the current research. Earlier work along these lines, on a learning system called T2 (Yee et al., 1990), has led to the approaches in this proposal which both generalize some of T2's techniques and address some of its anticipated scaling problems. The weak learning approaches currently proposed should remain compatible with, and complement, knowledge-based approaches such as those used in T2. I hope that eventually the two lines of research can be merged into a single approach enabling an agent facing a specific task to bootstrap itself from weak beginnings to the use of strong learning techniques.

1.2 Evaluation and abstraction

The proposed research on forming abstractions is grounded in a mathematical theory for optimal control that is well-established in engineering fields such as control theory and operations research. Tasks are posed as *Markov decision processes* which can be optimized, in principle, via the technique of *dynamic programming* (DP). This theory along with recently developed learning methods based on DP are presented in Section 2. This framework for learning and control encompasses many AI issues that have traditionally been studied in relatively isolated areas, foremost among which are robotics, planning, and learning.

Central in this framework is the problem of learning to predict the long-term effects of control actions executed from various system states—without entailing the costs of look-ahead search. DP-based methods can learn predictions in the form of a *value function* which assigns to states, numeric values reflecting the achievability of both immediate and long-term performance goals following from the states. Although in many circumstances DP-based methods provably converge to state values reflecting optimal performance in a task, the methods generally have poor scaling properties. Learning the optimal values of states remains unacceptably slow for tasks with numerous states and actions or with long delays between actions and their most significant outcomes.

The proposed research seeks to enhance the scaling properties of DP-based learning methods by focusing on the representations used to implement and learn value functions. I present two approaches to forming abstract representations of the states of a dynamical system, given goals for controlling the system. The first approach uses abstract classes of values to induce abstract classes of states. The second approach constructs hierarchical connectionist networks composed of Boolean units. The aim is to develop units acting as abstract features or concepts that represent states at multiple levels of abstraction. Learning based on abstract classes of values and states is expected to improve the efficiency of generalization as well as the determination of actions for performance.

Some assumptions about tasks are made to support the research on forming abstractions. First, tasks are not "all or nothing," i.e., they are simplifiable in useful ways, leading to a range of possible performance levels. Note that even win-lose tasks such as chess recognize multiple levels of skill. Second, abstractions for achieving the various performance levels are definable directly from the given task representations. Although not strictly necessary, this allows one to avoid the significant problem of uncovering *hidden state* information. Finally, task experiences are sufficiently redundant that abstractions can be built up in a statistical fashion (at least initially). This provides weak learning methods with sufficient information about the specific task being performed.

The proposed research is consistent with the view that a significant difference between novices and experts lies in their representations of tasks. Novices tend to employ structural descriptions of tasks whereas experts employ functional ones composed of features and concepts that explicitly relate structure to function in a task. This difference stems from basic computational economics. Computations employing meaningful features and concepts should lead to important inferences much more efficiently than computations based on unanalyzed structural representations, which often entail combinatorial complexity. The costs of forming, storing, and accessing functional representations are thus justified by the increased effectiveness of learning and performance in a given task.

Without special constraints, it is impossible to determine *a priori* what information about a task's structure constitutes key functional features and concepts and what constitutes minor detail. Such distinctions depend both on the task goals and on the dynamics of the controlled system. Significant features and concepts can thus only be determined by regressing goals through system dynamics. Intelligent agents must be capable of learning meaningful and effective task representations for themselves, and they must be able to put such knowledge to practical use both in directly improving performance and in improving their ability to do further learning.

Overview This proposal has two main parts. The first part, comprising sections 2, 3, and 4, presents the context, issues, and guidelines for the research. Section 2 describes the control and learning framework of *Markov decision processes* and *dynamic programming*. The poor scaling properties of DP-based learning methods will be addressed by using *function approximation* to implement *evaluation functions*. Section 3 focuses on function approximation and the special concerns of the control framework. Section 4 discusses the issue of *exploration* in DP-based learning.

The second part of the proposal, comprising sections 5, 6, and 7, presents two approaches for the efficient use of memory resources in DP-based learning. Section 5 first briefly describes the T2 system which is a partial basis for the two approaches. Section 6 proposes using a hierarchy of abstract classes of values to induce abstract classes of states for a task. Section 7 proposes learning the abstract state classes by constructing hierarchical connectionist networks. This approach is based on an algorithm, called the *address algorithm*, for training linear threshold units having variable-sized receptive fields.

2 A Framework for Control and Learning

This section describes the theory of Markov decision processes and learning methods based on dynamic programming (DP). It primarily follows the treatments in (Barto et al., 1990; Watkins, 1989; Dean & Wellman, 1991; Ross, 1983). For presentations of Markov decision processes (henceforth *tasks*) and DP from the mathematics and engineering literatures see (Bellman, 1957; Howard, 1960; Bellman & Dreyfus, 1962; Bellman, 1971; Ross, 1983; Bertsekas, 1987). Recent AI research has begun to bridge the gap between analytic theories of control and the forms of intelligent behavior and computational approaches studied in AI and other cognitive sciences. The machine learning and *reinforcement learning* methods presented in (Samuel, 1959; Samuel, 1967; Barto et al., 1983; Sutton, 1984; Sutton, 1988; Watkins, 1989; Kaelbling, 1990; Sutton, 1990) are related to Markov decision tasks and the technique of DP. Integration of control theory and AI research in learning, planning, and heuristic search is discussed in (Dean & Wellman, 1991; Miller et al., 1990; Barto et al., 1991; Barto, 1990; Barto, 1991; Sutton, 1991).

2.1 Markov decision tasks

Markov decision tasks are formalized dynamical systems and performance measures. Performance measures are procedures for evaluating the outputs of a system over time. System outputs can be influenced through inputs called *control actions*, and any method for selecting actions is called a *control policy*[1]. The goal of a task is to acquire an executable policy that is optimal with respect to a given performance measure. Before considering how optimal policies are obtained, we define systems and performance measures.

System dynamics: states and actions A dynamical system is represented by a set of states X, a set of actions A, a set of state transitions, i.e., ordered pairs of states, and for stochastic systems, a set of transition probabilities. For simplicity, we consider only discrete-time formulations of tasks comprising finite numbers of states and actions. Tasks commonly represented with continuous-valued variables can be formulated by using discrete, finite approximations of the continuous values. The learning of abstractions

[1]Policies are also called *control laws* or *strategies*.

is expected to help compensate for the poor tradeoff between accuracy and computational complexity which can afflict such discretization approaches. Nevertheless, the representation of continuous values could arise as a secondary research issue.

Given finite states and actions, a system can be represented as a stochastic *finite state automaton* (FSA) whose states are the system states and whose transition arcs are labeled by actions and transition probabilities. A state might disallow particular actions, a condition represented by the lack of transition arcs labeled with the disallowed actions. Let $A(x) \subseteq A$ denote the set of actions available in state $x \in X$. When a system in state x receives input action $a \in A(x)$, an event denoted $\langle x, a \rangle$, the system transits to a new state $y \in X$ via a process that may be stochastic. That is, $\langle x, a \rangle$ may generate y according to a fixed probability distribution associated with $\langle x, a \rangle$. Let $P_{xy}(a)$ denote the probability that $\langle x, a \rangle$ yields state y in one time-step. In the FSA representation of a deterministic system, each state-action $\langle x, a \rangle$ corresponds to a single transition arc emanating from x. In a stochastic system, $\langle x, a \rangle$ corresponds to a set of arcs from x to each state $y \in X$, where each arc is further associated with a transition probability $P_{xy}(a)$.

System dynamics constrain trajectories through the state space X. Usually, many trajectories are either impossible or have a very low probability of occurrence. This is represented in an FSA by missing arcs or arcs associated with low transition probabilities. System dynamics limit a controller's ability to navigate through the state space.

The preceding formulation implies that system dynamics satisfy the *Markov property* which holds that state transition probabilities depend solely on a single state-action pair $\langle x, a \rangle$. This means that at any given time, the future behavior of a system depends only on the current state and the action taken. All information about the system and its history that could affect future behavior is reflected in each state description, or, equivalently, any information not represented by a state cannot affect behavior.

This strong condition on the formulation of systems significantly simplifies the optimal control problem because in selecting an action, one need only consider the current state, without regard to previously visited states. Unfortunately, readily available information about a system, e.g., observable structures, might not yield Markovian state descriptions. Such information, sometimes called *sensations*, might fail to distinguish between two or more

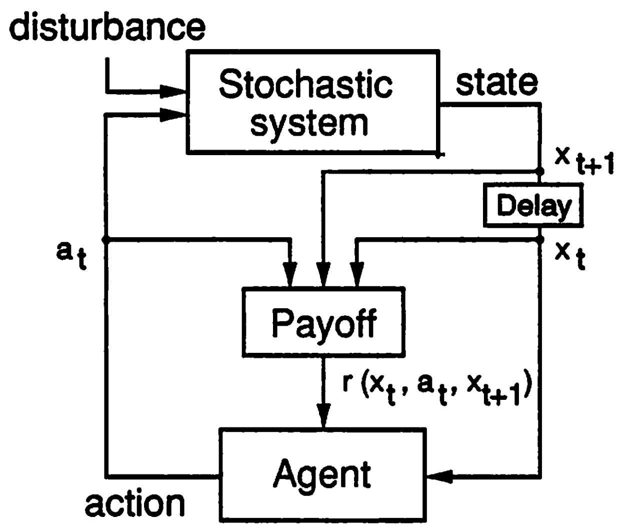

Figure 2: Learning control from payoffs. The agent receives numeric payoffs as a function of its actions and the corresponding state transitions of the system. The control objective is to maximize a function of all payoffs received across a time horizon.

true system states, a condition which could lead to unaccountable system behavior. In FSA terms, sensations correspond to state labels, which need not be unique. Common approaches for constructing Markovian state descriptions from non-Markovian sensations include maintaining a history of past sensations, e.g., using a *tapped delay line*, or using methods for uncovering hidden state information (Rivest & Schapire, 1987; Schapire, 1988; Mozer & Bachrach, 1991; Whitehead & Ballard, 1991). Although coping with hidden state is an important and interesting issue, it is not a focus of the proposed research; Markovian state descriptions are assumed.

Task performance: payoffs, horizons, and values A performance measure for a task consists of numeric *payoffs*, associated with state transitions, and a *discount horizon*, which specifies how payoffs are integrated over time. Each time an agent executes an action it receives a payoff [2] which is

[2]Payoffs are also called *rewards* (possibly also with *penalties*), or *costs*, which are often restricted to non-negative values but carry a negative connotation.

a bounded, real-valued scalar. In general, a payoff depends upon the state, the action, and the resulting state associated with a transition. Figure 2, adapted from (Barto, 1991), shows a payoff component added to the basic control architecture of Figure 1. We shall assume that payoffs are *stationary*, i.e., that they do not depend on time (unless states explicitly represent time). Because payoffs may be generated stochastically from actions, one is usually most concerned with the *expected payoff* of executing an action a in a state x; this is denoted $R(x, a)$.

Given feedback about the state of a system at each time step, policies specify actions. The performance of a system controlled by policy π is measured by a *value function* V^π which maps states into real values, $V^\pi : X \rightarrow \Re$. The value $V^\pi(x)$ of state x is computed from a discount horizon and the sequences of payoffs received by starting in x and executing π.[3] We consider here only values determined from weighted sums of payoffs.

A common and simple type of discount horizon will be used in the current research: a geometric sequence of the form

$$\left(1, \gamma, \gamma^2, \ldots\right), \quad 0 \leq \gamma < 1.$$

The value of any state x is then defined as

$$V^\pi(x) \triangleq E_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid x_0 = x\right], \tag{1}$$

where x_t and r_t denote, respectively, the state and ensuing payoff at time t, and E_π is the expectation operator given that all actions are selected by π. The value $V^\pi(x)$ is the infinite sum of discounted payoffs that can be expected to accrue by starting in state x and following policy π forever. Although geometric horizons are infinitely long, the effects of future payoffs diminish exponentially in time due to the γ^t discount factor. The time required for future payoffs to become negligible depends upon the magnitude of γ which thus controls the effective "length" of a geometric horizon.

Note that geometric discounting changes the *scale* of future payoffs but not their relative magnitudes (Berry & Fristedt, 1985). This fact together

[3]By analogy with animal learning, payoffs can be viewed as *primary reinforcements* and values as *secondary* ones. A state's value is a prediction of achievable primary reinforcement.

with stationary payoffs and the Markov property ensure the existence of a stationary, deterministic policy that is *optimal* in a sense defined below (e.g., see Ross, 1983). Thus, one need only consider policies π that are functions of states: $\pi : X \rightarrow A$.

Q-values Instead of associating values with states, it sometimes proves useful to associate them with state-actions $\langle x, a \rangle$ (Watkins, 1989). Because the discount horizon is a geometric sequence, Equation (1) can be rewritten in the recursive form

$$V^{\pi}(x) \quad = \quad R(x, a_0) + \gamma \sum_{y \in X} P_{xy}(a_0)\, V^{\pi}(y), \qquad (2)$$

where a_0 is the action selected by π when $x_0 = x$. This equation shows that the value of x is the sum of the expected immediate *payoff* of state-action $\langle x, a_0 \rangle$ and the discounted *value* expected from succeeding states. Instead of requiring the first action a_0 to be the one selected by π, following Watkins, Equation (2) can be generalized by specifying the choice of action at time $t = 0$ as a parameter a:

$$\begin{aligned} Q^{\pi}(x, a) \quad &= \quad R(x, a) + \gamma \sum_{y \in X} P_{xy}(a)\, V^{\pi}(y) \\ &= \quad R(x, a) + \gamma \sum_{y \in X} P_{xy}(a)\, Q^{\pi}(y, \pi(y)) \qquad (3) \end{aligned}$$

State-action values, called *Q-values*, provide for arbitrary initial actions, but all subsequent actions are selected by π. This proposal discusses the problem of learning state values with the understanding that, except as noted, the generalization to learning Q-values is also intended.

Optimal policies Both state values and Q-values measure the performance of policies over a given time horizon. An *optimal policy* π^* yields values greater than or equal to those of any other policy. That is, π^* is optimal if and only if for all policies π and all states $x \in X$, $V^{\pi^*}(x) \geq V^{\pi}(x)$.[4] These maximal state values—the *optimal values* of states—are unique, but

[4]If payoffs are nonnegative costs, then optimal policies *minimize* values. The existence of optimal policies requires a proof, which is not supplied here; see, e.g., (Ross, 1983) or (Bertsekas, 1987).

there may be different ways of attaining them, i.e., if policies π^*_1 and π^*_2 are both optimal, then $V^{\pi^*_1} = V^{\pi^*_2} \triangleq V^*$. Thus for each Markov decision task, there is a unique *optimal value function* V^*.

Payoffs and discount horizons together implicitly define V^* which, in turn, defines the control objectives of a task. Such simple goal structures undoubtedly fail to encompass the complexity of biological agents' representations of goals which surely interact with other problem-solving representations. The ramifications of current simplifications are difficult to anticipate, however, because the theory of Markov decision tasks is so general. Many tasks with complex goal structures (e.g., animal foraging) can nonetheless be formulated. Further research within this framework should help clarify the inadequacies of current formulations of task goals (cf. Singh, 1991).

2.2 Learning optimal values via dynamic programming

In general, long-term optimal control of dynamical systems requires an exhaustive search of the space of potential policies—a space which is exponentially large in effective horizon length. In practice, one tries to exploit structures in tasks to eliminate most suboptimal policies from consideration. Unfortunately, many tasks lack sufficient structure for making *a priori* assessments of the long-term performance of policies. One approach is to use methods for discovering such structures on an individual-task basis. Another related approach is to use heuristics applicable to subclasses of tasks although this usually entails giving up guarantees of optimality. Both of these approaches are used in the proposed research.

The foundation for current proposals is the technique of dynamic programming. DP exploits the Markov property of tasks to eliminate vast numbers of suboptimal policies from consideration by systematically organizing the optimization process. DP is based on Bellman's *Principle of Optimality* (Bellman, 1957, p. 83) which essentially holds that optimal policies can be found by first optimizing single decisions and then working "backwards," optimizing successively earlier decisions in stages. DP is thus a recursive process: each stage entails only a *single-step* optimization search, given the results compiled from the previous stage. This procedure, which is orders of magnitude more efficient that an unconstrained search, is based on the

Markov property of states: one can incrementally optimize in a backwards fashion because preceding decisions can be safely ignored given the current state. This is not true, of course, in the forward direction: current decisions may surely depend upon future possibilities.

Whereas payoffs reflect only the immediate costs and benefits of actions, values accurately measure long-term performance. The intermediate results of DP's shallow optimization searches are compiled into value functions V_i which form a sequence leading ultimately to the optimal value function V^*. One version of this process, called ***value iteration*** (e.g., Howard, 1960), is characterized by ***greedily backing up*** values:

$$V_{i+1}(x) \quad = \quad \max_{a\in A(x)} \left[R(x,a) + \gamma \sum_y P_{xy}(a)\, V_i(y) \right], \qquad (4)$$

where V_i is the i^{th} approximation of V^*. This step estimates the value of a state x by finding an action a that maximizes the expected immediate payoff $R(x,a)$ and the expected estimated value of subsequent states. The value estimates of all states other than x are unchanged. This process is guaranteed to converge, $\lim_{i\to\infty} V_i = V^*$, provided the estimates of all states are backed up sufficiently often (Bertsekas & Tsitsiklis, 1989; Barto et al., 1991).

V^* leads to optimal policies in a straightforward manner. A policy π^* is optimal if and only if

$$\forall x,\ \pi^*(x) \quad = \quad \arg\max_{a\in A(x)} \left[R(x,a) + \gamma \sum_y P_{xy}(a)\, V^*(y) \right].$$

Any policy that selects actions greedily with respect to both immediate payoffs and V^* is optimal.

Value iteration methods are the basic learning approaches employed in the proposed research. Another DP approach, ***policy iteration*** (Howard, 1960), involves the storage of intermediate policies π_i in addition to the value functions V_i. This raises issues regarding the interactions between value functions and policies which, again, are important but are not the focus of current research. Despite the advantages of value iteration over unconstrained searches for optimal policies, the propagation of values occurs only on a state-by-state basis, making this basic form of the approach impractical in many cases. One therefore considers combining DP-based learning methods with methods for efficiently propagating value information throughout the state space of a task. Forming a basis for the efficient generalization of values is the central issue addressed by the formation of abstract representations for tasks.

3 Function Approximation in Control

An important motivation for the proposed research is the view that DP-based methods can be scaled up by incorporating function approximation techniques into the learning of the optimal value function V^* (or Q^*). Function approximation is a ubiquitous problem that has been extensively studied in numerous forms. The formulations most relevant to the current research come from the areas of machine learning and connectionism in AI, which, in turn, draw upon areas of mathematics and engineering such as *statistical pattern recognition*, *regularization theory*, and *digital signal processing*.

In function approximation, information about a *target function* f^* is usually supplied as a set or sequence of supervised training examples of the form: $\langle x, f^*(x) \rangle$, where x is an element of the function domain and $f^*(x)$ is the *target value* of x. Training examples are combined with prior knowledge or biases regarding f^* to produce an approximating function f, which should be close to f^* according to some distance measure for functions such as the L_2 norm. The approximation f is a member of a class of functions which is determined by the approximation method; the class might or might not contain f^*.

Incorporating function approximation into the DP-learning framework is not as straightforward as might first appear. For example, if V^* is not within an approximator's function class, then what would constitute a desirable approximation? Common answers come to mind, e.g., an approximation that minimizes a standard error measure such as the sum of squared errors over the training set or the difference between the approximation and V^* according to the L_2 norm. Although such criteria seem natural in function approximation, they are not necessarily the best ones for the control objective of finding an optimal policy with least effort. Indeed, it is seems that any criteria regarding the appropriate approximation of V^* should depend upon its role in the derivation of the control policy: different derivation procedures might place different demands on the approximation. An obvious goal is that an optimal policy should be derivable from an approximation of V^*, even if V^* itself is not implementable. However, it is not obvious how to translate this goal into criteria governing the approximation of V^*.

In the proposed research, I will avoid the problem of defining such approximation criteria by assuming that the class of approximating functions contains V^*. This could be achieved either through knowledge that V^* lies within a specific function class or through the use of a *universal approximator.* I will follow the latter approach: the approximators used will, for practical purposes, be capable of implementing any possible optimal value function.

3.1 Generalization, bias, and representation

One key to the efficient learning of V^* is the *generalization* of value estimates. Instead of learning values on state-by-state basis, generalizations amplify the information provided by a single backup by automatically propagating related value information to sets of states. The accuracy and efficiency of generalizations depend heavily on *inductive biases* (Mitchell, 1980)—cf. *regularizers* (Poggio & Girosi, 1990)—which are constraints determining which states are affected and how their value estimates are modified. *Strong* biases are precise, task-specific constraints from which one can obtain very broad, highly accurate generalizations of single training examples. *Weak* biases are general functional constraints, such as *smoothness*, which are applicable to many otherwise diverse functions. Weak biases do not reliably lead to efficient learning: only a large amount of training data can guarantee the accuracy of an approximation to a specific target function. The ability to generalize reliably, and hence learn efficiently, requires strong biases appropriate to the specific task being addressed.

Unfortunately, as noted, the class of Markov decision tasks lacks sufficient structure for deriving strong *a priori* constraints on value functions. In specific cases, one can supply prior knowledge by hand, e.g., using techniques such as modeling, explanation-based generalization, gradient-estimation, nominal functions, or shaping. But such solutions can be costly, requiring substantial engineering effort whose results are often narrow in application, and there is often little guarantee of success. Alternatively, one can use weak methods or rely on heuristics that may fail drastically in some cases, either of which has obvious drawbacks. Finally, one can use methods that start out weak and become strong during learning. That is, one can use methods that *acquire* strong inductive biases (cf. Sutton, 1992).

Change of representation and inductive bias The learning of abstractions yields generalization biases through *change of representation.* The modification of inductive bias through change of representation or *constructive induction* has been studied in many forms in machine learning (e.g., Rendell, 1985; Utgoff, 1986; Schlimmer & Granger, 1986; Benjamin et al., 1988; Matheus, 1991). Direct comparisons between such approaches and the approaches proposed below are somewhat difficult to make. Most previous approaches assume a supervised learning paradigm rather than the reinforcement/optimization paradigm currently assumed. Also, much of the work is formulated in terms of symbolic representations and logical operations, whereas I propose using function approximators that are closely related to connectionist approaches to constructing *distributed, subsymbolic* representations (Rumelhart et al., 1986; Smolensky, 1988; Hinton, 1989; Hinton, 1990).

The proposed research shares with AI projects such as SOAR (Laird et al., 1986) and PRODIGY (Minton et al., 1989) an emphasis on learning within a general problem-solving framework, however, the current framework is based on a theory of optimal control developed by branches of mathematics and engineering independent of AI. The theory gives a formal characterization of dynamical systems and control processes, which although it was not specifically designed for them, appears to encompass many cognitive tasks. Previous work has investigated the integration of reinforcement learning with this control framework (Watkins, 1989; Barto et al., 1990; Barto et al., 1991; Sutton, 1991). The approaches in this proposal aim to bolster weak reinforcement learning methods used in this control context by forming abstractions that provide task-specific inductive biases.

Representational bias The representation of inputs is a crucial form of bias. Countless learning approaches rely on *representational similarity* as the basis of generalization. Inputs are assumed to have similar function values if and only if they look similar. Inputs dissimilar in appearance are often treated as functionally unrelated. Similarity means *proximity* in the representation space of the function domain, as measured by a metric defined on the space. Typically, a standard metric is chosen such as the Hamming metric or Euclidean distance. The name *similarity-based learning* (SBL) is applied to this class of approaches.

Standard similarity metrics are weak, i.e., they are not specialized for a particular domain or task. For a specific application to have a hope of success, the designer of representations for a domain must at least ensure that inputs having similar representations also have similar values. That is, *form should not conceal function.* This is a merely a necessary condition without which SBL generalizations have no basis.

Practical success usually requires an additional, stronger condition: form should *reveal* function. That is, one should strive to design input representations that reflect, as explicitly as possible, all and only the features of inputs that determine their functional behavior. Such "functional representations" encode task-specific biases, and they enable simple syntactic similarity metrics to produce accurate generalizations for a task. Such representations can thus lead to efficient learning, but, far from being weak, significant task-specific knowledge, and often ingenuity, is required to craft functional representations appropriate for weak metrics.

For many tasks, the costs of designing representations are hidden because traditional representations are compatible with standard generalization metrics. For example, in areas such as robotics where interactions with physical systems are a primary concern, research relies on extensive knowledge of physics, mechanics, and continuous mathematics. Thus, highly engineered representations already exist or are easy to derive for some tasks. It is noteworthy that such "natural" representations may have taken centuries to develop.

Research into the computational aspects of phenomena such as perception, language, and problem-solving, generally lacks the advantage of such highly engineered representations. How, for example, should one gauge similarity among items such as samples of speech, journal articles, or chess boards—objects whose traditional structural representations are seldom compatible standard generalization metrics? Even in domains such as robotics, standard metrics might only be partially effective: due to a lack of appropriate structure, there often remains substantial room for improving traditional representations in specific cases.

Common input representations are often an inadequate basis for learning, especially if the representations reflect only the structure of inputs (Saxena, 1991a; Saxena, 1991b). Good representations for a task must explicitly reflect functional differences between inputs, differences which depend on system dynamics (e.g., Dayan, 1991). Good *abstractions* further reflect the importance of such differences relative to task goals. Abstract representations are not easily formed in general, but they are necessary for efficient learning and performance.

3.2 Constraints on control learning

There are numerous possible methods for approximating a value function. The choices can be narrowed somewhat by considering the constraints on learning imposed by the control setting and the use of DP-based methods. This section examines some special concerns which, taken together, have received relatively little attention in learning research on function approximation.

For the following discussion, it will be useful to characterize approximation methods according to a spectrum whose extremes I call *local* and *global*, as an indication of generalization behavior. Roughly speaking, in response to an individual training example, a local method radically changes its approximation f in a small neighborhood of the input, as defined by a similarity metric. A global method produces an incremental change in f across a broad region of the input domain. The prototypical local method is a lookup table; for global methods, a prototype is the estimation of a constant value over a domain, e.g., an average; another common global method is estimation of a linear surface.

Table 1 lists a number of properties that are often, but not always, displayed by methods of each type. Local methods store and recall specific training data quickly and accurately but tend to produce poor generalizations for novel regions of the input space. They can implement complex functions but are correspondingly sensitive to noise in the data. Global methods have complementary strengths and weaknesses. They can generalize well but can be slow in memorizing specific data. They tend to be noise-resistant but may only be able to implement simple functions. Finally, unlike most global methods, local methods can sometimes distinguish between "recall"

Generalization type:	Local	Global
Prior knowledge	weak	strong
Fn. complexity	high	low
Noise handling	overfits	resists
Memory:		
size	large	small
storage	fast	slow
recall	fast (?)	fast (?)

Table 1: Stereotypic properities of approximation methods: local vs. global.

and "prediction," i.e., between output estimates supported by much training data and estimates for which the training data is sparse (Moore, 1991; cf. Kanerva, 1988).

Bounded generalization errors An important constraint on function approximation in DP-based learning derives from the iterative process of backing up values. Proofs that DP methods converge to V^* assume that in backing up the estimated value of a state, all other estimates remain unchanged. On the contrary, approximation seeks learning efficiency through generalization: the value estimates derived from specific experiences should be generalized to a wide range of similar states (or state-actions). Generalization errors, however, could thwart the convergence of the iterative backing up process, making V^* unachievable even if it is within an approximator's function class (Barto et al., 1991). If generalizations are sufficiently accurate, of course, then there is no problem. Specifically, because convergence to V^* depends only on the *maximum error* in the value estimate of any state, if generalization errors never exceed this maximum, then the guarantee of convergence to V^* is maintained.

Unfortunately, ensuring bounds on the magnitudes of generalization errors would seem to require considerable knowledge of V^*, knowledge which would be unavailable in general. Constraining generalization so as to guarantee convergence to an appropriate approximation of V^* might be the most formidable approximation problem in DP-based learning. Approaches for addressing this problem are outlined later in this section, but they are not definitive solutions.

Non-stationary training targets Learning a value function through DP iterations is not a supervised task because there is no *teacher* to provide the true target values of V^*. Training examples consist of an input state x and its value estimate which is recursively based, in part, on the current approximation of V^*, denoted V_i (see Equation 4). Because the value estimates V_i are continually changing, the "target values" of training backups are non-stationary. Presumably, value estimates eventually converge to the true optimal values, but before they do, they may be subject to a great deal of noise from inaccurate estimates in the V_i functions. An approximation method must therefore track the moving target values while simultaneously trying to filter out the noise—not a simple feat. Local methods can track changing values but can just as easily track noise. Global methods can filter out noise but can also be slow to adapt to moving targets, thereby negating the benefits of generalization.

Dependent training inputs Another difficulty for learning in a control framework is the fact that the training inputs, states x_t, are not generated as uniform samples from a fixed distribution, an assumption of many learning methods (e.g., Valiant, 1984). On the contrary, due to system dynamics, new input states are highly dependent on recently visited ones. Global methods that "forget" data can have problems with dependent inputs if the sampling rate from various regions of the input space is too low. Such methods will tend to track recent inputs with all of their resources, replacing information previously learned in one region with information from the current region. This can be a problem, for example, in connectionist networks (Sutton, 1986) where the phenomenon is called *temporal crosstalk* (Jacobs, 1990).

If an adequate system model is available, then a possible way to alleviate input dependence is to generate independent training data off-line from the model (cf. Sutton, 1990). Otherwise, one requires a learning method that can concentrate on circumscribed regions of the state space without unduly disrupting the learning conducted in other regions at other times. Local methods offer this property at the price of poor generalization across regions of the state space. A more desirable solution would efficiently bias generalizations of new data without sacrificing what has been learned in regions removed from the current context.

3.3 Approach: memorize and consolidate

Control and DP constraints on learning suggest that approaches combining aspects of both local and global approximation are required. The ultimate goal of any method should be to form the most parsimonious possible description of a task's optimal value function because such a description is the most likely to produce correct value estimates for future inputs. Formal justifications for this principle of *Occam's razor* are derived from *algorithmic information theory*, *computational learning theory* (Blumer et al., 1987), and *statistical estimation*. See (Saxena, 1991b; Saxena, 1991a) for explication and references.

Concise approximations are often achieved through the imposition of strong constraints, but such approaches are problematic for DP-based learning, particularly with regard to bounding generalization errors. The DP framework calls for weak approaches that have local generalization behavior, but such approaches usually do not lead to concise function descriptions and do not scale well to large tasks. The goals and constraints presented here thus argue for learning methods that accelerate. Learning should begin weakly with very local generalization behavior, but biases should strengthen through experience, leading eventually to large-scale generalization and concise function descriptions.

This behavior can be attained through the management of *approximation resources*, which are structures such as terms in Boolean formulae, terms and coefficients in polynomials, nodes in decision trees, memory locations in hash tables, or units and weights in connectionist networks. Such structures are essentially memory resources used for storing and propagating information. Methods for DP-learning should begin with abundant, weakly-constrained memory resources that are initially allocated liberally as training data is received. Such an approach is essentially *open hashing*, i.e., the implementation of a virtual lookup table covering the space of inputs.

Such memory-intensive approximation methods have been used in robotics control tasks (Moore, 1990; Atkeson & Reinkensmeyer, 1990). One popular type of approach, which is particularly close to lookup table implementations, is based on the CMAC approximator (Albus, 1975b; Albus, 1975a) and its variations (Moody, 1989; Lane et al., 1991; see Dean & Wellman, 1991, secs. 9.4 & 9.7, for discussion and further references). Another popular type is based on *k-d trees* (Bentley, 1975; Omohundro, 1987). Memory-based approaches usually use standard similarity metrics to produce local generalizations, the effectiveness of which relies on the appropriateness of input representations for their tasks. Such approaches also tend to rely on large amounts of training data. If generalization biases are adapted at all, it is usually through the adjustment of the similarity metric rather than the input representations (e.g., Stanfill & Waltz, 1986). Current proposals do not require that tasks be given input representations compatible with common forms of similarity metrics. Change of representation offers great flexibility and power for improving similarity-based generalization across a diverse range of domains.

Efficiency requires that, as learning progresses, strong generalization biases are acquired. A general approach is to increasingly constrain resources, forcing fewer and fewer resources to implement larger and larger portions of the approximation. Resources can be constrained by limiting the number and ranges of free parameters, e.g., by eliminating terms, nodes, units, or weights (e.g., Alpaydin, 1991; Mozer & Smolensky, 1989; Le Cun et al., 1990) or by restricting parameter values (e.g., Weigend et al., 1991; Nowlan & Hinton, To appear).

Handling learning constraints Strengthening initially weak biases during learning addresses several issues in the DP framework. Although local methods do not guarantee bounds on generalization errors, one expects them to be small typically, and in any case such errors will be highly localized in the state space during the early stages of learning. This should allow for recovery from errors for commonly visited states. As subsequent learning leads to generalizations at larger scales, any types of states receiving significant generalization errors should have only low probabilities of being visited. Furthermore, because control improves as learning progresses, the diversity of state trajectories should shrink, thus further removing an agent from encounters with states having inaccurate value estimates. Localizing initial generalizations should thus allow the DP backup process to converge to an approximation of V^* that is highly accurate for the types of states likely to be visited during control.

Constraining resources should also help track non-stationary targets and remove noisy estimates because eliminating allocated resources amounts to a form of forgetting. Value estimates that are "out of line" with current information because they are old or have been corrupted by noise (e.g., through generalization errors), will be replaced with more current estimates as the resources used to represent the eccentric values are removed. This may also be viewed as a process of smoothing: reducing resources reduces the complexity of the functions that can be implemented.

Finally, beginning with a highly local method helps ensure that input dependencies will not lead to temporal crosstalk, i.e., the usurpation of approximation resources allocated to regions of the state space not recently visited. Subsequent deallocation of resources should strive to ensure that some remaining resources still approximate V^* in the affected regions.

Accommodating function complexity Although the goal of constraining resources is to produce concise descriptions of V^*, an important advantage of local methods is their ability to implement highly complex nonlinear functions. This ability is especially needed when given state representations lack explicit functional features. In general, purely structural state representations lead to complex optimal value functions because information about task goals and system dynamics is absent. Therefore, it is important to retain sufficient approximation resources to cover important regions of the state space where V^* has high complexity.

Not all approximation errors need be treated equally. Accurate value estimates are most important for frequently visited regions of the state space and for states affecting critical action selections. An action selection is critical if different choices produce great differences in performance, i.e., if V^* has a steep gradient with respect to the available actions. It makes sense from the "economic" standpoint of maximizing control performance to allocate necessary resources for the approximation of V^* for important states, at the possible cost of losing accuracy in less important regions of the state space.

Sections 6 and 7 discuss specific approaches for implementing approximation methods with the properties outlined here. The basis for the approaches is the use of abstract, hierarchical representations of values and states. Hierarchies allow one to allocate representational resources in an intelligent manner: increasing resources for important regions with high complexity, restricting resources where they are unnecessary either because V^* is well-behaved or because the regions in question are relatively less important. As resources become increasingly restricted, one may consider that an approximation method is strengthening, i.e., that it is incorporating a stronger generalization bias for the specific task. This can also be viewed as abstracting the task by transferring episodic information about individual states (state-actions), into semantic "rules" for determining values for classes or types of states.

4 Exploration for Optimization

In addition to constraints on learning, optimal control introduces the goal of *exploration.* Recall that optimal state trajectories for a task are not provided to an agent, which must consequently search through the space of policies. DP implicitly organizes this search based on the Principle of Optimality, but practical limitations on knowledge and computational resources often force one to approximate the DP process. Consequently, in considering the efficiency of function approximation methods, the issue of exploration must not be overlooked.

In fact, there are at least two types of exploration problems in optimal control. One type, which is not of concern here, seeks information about the state transitions of a system or the payoffs of a task. By executing actions whose immediate effects are imperfectly known, one can build or improve *models* of system dynamics or payoffs (e.g., Barto & Singh, 1991; Schmidhuber, 1991b; Thrun & Möller, To appear; see Thrun, To appear, for an overview and further references). A second type of exploration, which is of current concern, seeks information about the optimal values of states. Even if complete, accurate models of dynamics and payoffs are available, one must explore the consequences of actions whose effects on *long-term performance* are not known with certainty. Such exploration is unavoidable if optimality is to be guaranteed. In practice, of course, one often settles for good performance which may be suboptimal. Unfortunately, practical limitations can also make it difficult to exploit the DP optimization process in its purest form.

Solving the DP recursion DP backups are recursive: the new value estimate for a state x depends on the current estimates of x's successors,

Current estimate $V_i(x)$	$\leftarrow$	Backed-up estimates $V_i(y)$	Result
good	$\leftarrow$	*good*	useless (usually)
bad	$\leftarrow$	*bad*	useless
good	$\leftarrow$	*bad*	destructive
bad	$\leftarrow$	*good*	useful

Table 2: Four types of value backups.

the $V_i(y)$ in Equation (4). Thus, like all recursive processes, DP implicitly proceeds by first solving the *base cases*, those cases with no recursive portion. "Base-case backups" are those for which the value estimates of succeeding states $V_i(y)$ are accurate with high certainty. For example, many tasks have *terminal states* (e.g., wins, losses, and draws in a game) whose values do not depend on any succeeding states. Tasks without terminal states may nevertheless have states whose optimal values are known with high accuracy: e.g., states might be associated with very large payoffs that are certain to overwhelm the values of succeeding states. In general, the more difficult it is to obtain base cases for a task, i.e., known optimal values for states, the more difficult it is to exploit DP to organize the optimization process.

DP recursion then proceeds "backwards" from the base cases, forming new value estimates from those whose recursive parts have been previously solved. This propagates the true values of V^* across the state space as quickly as possible. Organizing backups in any other fashion could lead to wasted effort. Table 2 characterizes four general types of backups. Replacing a good optimal value estimate with a good one or replacing a bad estimate with a bad one, makes virtually no progress in determining V^*, and replacing a good estimate with a bad one does obvious harm. Only the replacement of a bad estimate with a good one is useful. Performing backups in the wrong order can negate DP's organization of the optimization process and may even compound the problem by forcing one to explore specific action sequences multiple times before accurate estimates of V^* are ever obtained.

Exploration: control of uncertainty Exploration for optimization is simply the problem of actually implementing DP backups in the manner implied by the Principle of Optimality. The goal is to maximize the frequency of useful backups and minimize, especially, the frequency of destructive ones. Determining the utility of a backup requires knowledge of the accuracies of value estimates, or, at least, knowledge of their relative accuracies. An approximation method capable of supplying such information could thus be very beneficial.

Two facets of the exploration problem are (a) obtaining information about the accuracies of value estimates and (b) propagating the information. The best source of accurate value estimates for a task usually comes from its definition which often includes setting fixed values for special states such as goal states or hazards. Such values define the "boundary conditions" of a task, and the DP process is used to propagate these optimal value constraints throughout the remaining state space. Heuristics might also be used to determine the accuracy of values. For example, if an agent has extensive experience in a region of state space (where regions are determined by system dynamics) and the values of various states within the region are stable, then there is reason to believe that the values are close to the optimal ones. Of course, there will always remain some chance that a future course of action could significantly alter the estimates.

Propagating the accuracies of estimates also has two aspects. The first is the mechanism by which such information is transferred during a backup. In addition to backing up the value estimates themselves, a mechanism is needed for backing up accuracy information. For example, one could keep information on variance together with the estimates (e.g., Kaelbling, 1990). Alternatively, one could maintain a separate function that gauges the accuracy of the value function estimates (cf. Schmidhuber, 1991a; Thrun & Möller, To appear).

The second aspect of propagating accuracy is tied to the issue of organizing backups, i.e., the issue of how the accuracy information is actually used to approximate DP. The exploration problem is exacerbated if it must be conducted on a real system, i.e., without the benefit of models of system dynamics and task payoffs. Actually executing exploratory actions, which generally include actions expected to lead to poor results, could seriously degrade control performance. This poses the classic tradeoff between control and estimation, i.e., between acting to optimize performance according to current estimates and acting to gain information that *might* improve those estimates (cf. Berry & Fristedt, 1985).

If sufficiently accurate models of transition dynamics and payoffs are available, exploratory actions can be simulated, leaving one free to execute on the real system the actions currently deemed best (cf. *certainty equivalence*). Additional benefits of models often include the ability to explore faster than real time and to explore in any region of the state space at any time (Sutton, 1990). Of course, attaining such models might itself entail significant costs, possibly including the first type of exploration conducted on a real system. If exploration of either type must be conducted on a real system, there might be a conservative tendency in executing exploratory actions, which might decrease the likelihood of finding truly optimal policies.

Exploration heuristics If a DP task provides sufficient structure, heuristic exploration strategies can improve learning efficiency. For example, Peng and Williams (Submitted) augment Sutton's (1990) *Dyna-Q* system, using a "surprise heuristic" to preferentially select backups for states whose predicted Q-values differ greatly from their backed up estimates. They show that their *Dyna-Q-queue* system, facing grid-world navigation tasks, converges to optimal performance faster using the surprise heuristic than using a simple random exploration strategy.[5]

Referring again to Table 2, the surprise heuristic suppresses *good-good* backups and *bad-bad* backups for which there is little "surprise," and it exploits special features of the task and learner. Destructive *bad-good* backups, which can also be surprising, seldom arise in Dyna-Q-queue because the initial Q-value function is zero everywhere, and there is no generalization: Q-functions are implemented by lookup tables. Thus, if a surprise occurs, it is most likely because a low-valued state should truly have a higher value. This structure also allows the surprise heuristic to eliminate many *bad-bad* backups for which bad predictions are consistent with bad backed-up estimates. Such cases are common in Dyna-Q-queue whereas, in general, *bad-bad* backups can also be surprising.

In structured learning situations, the use of appropriate exploration heuristics (e.g., see also Kaelbling, 1990; Moore, 1990, 1991) can yield substantial

[5]The surprise strategy is applied only to a partially developed model of a grid-world environment. On the real environment, random exploration alone is used both for developing the partial model and for additional Q-learning.

improvements over more expedient strategies such as introducing random noise into actions. Although heuristics inappropriately applied could be far worse than random strategies, it is possible that many tasks of practical interest are compatible with straightforward exploration strategies. Finally, note that if certainty estimates are available for a function approximator, then *good-good* backups can also be useful, if they change an uncertain estimate into a certain one. Due to the problem of exploration, *knowledge* about the accuracies of estimates is almost as important as the estimates themselves—a point rarely stressed in learning research on function approximation. The analysis presented here emphasizes both the difficulty of the exploration problem in general and the gains that can be achieved by considering structured subclasses of Markov decision tasks.

5 Features of the T2 System

We have seen how a class of optimal control tasks can be formalized as Markov decision tasks and how they can be solved in principle via dynamic programming, which forms the optimal value functions of tasks. We have examined issues in approximating optimal value functions, focusing on special learning constraints, generalization, representation, and exploration. From weak beginnings, agents should be capable of automatically acquiring strong generalization biases appropriate for their specific tasks. This proposal focuses on the acquisition of representational biases through the formation of abstractions. In particular, learned abstractions are expected to significantly improve the efficiency of DP-based learning methods. Sections 6 and 7 each propose approaches addressing these goals. Because significant aspects of the approaches originated from earlier work on a game-playing system, T2, this section briefly discusses the relevant aspects of that work as background.

The T2 system Precursors of some of the proposals presented in Sections 6 and 7 were implemented in a learning system called T2 (Yee et al., 1990), which formed concepts and features for evaluating states arising in two-person games. T2 combined a DP-based method similar to the one used in Samuel's checker-playing system (Samuel, 1959) with more recently developed generalization methods from machine learning, namely *explanation-based generalization* (EBG) (Mitchell et al., 1986) and *goal regression*

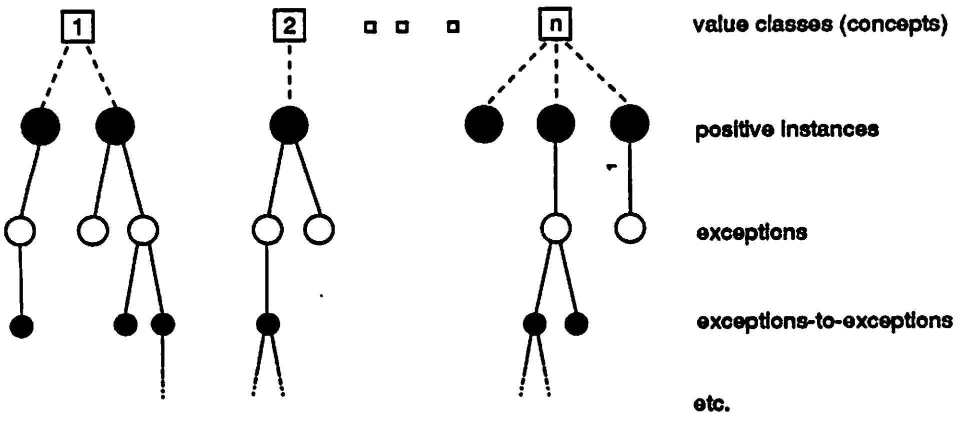

Figure 3: The structure of value classes in T2. Each value class contains states having a specific value. Each class definition is composed of Boolean features, structured as multilevel counterfactuals.

(Waldinger, 1976). Samuel first used a lookup table to approximate the value function (he later used polynomials whose terms corresponded to hand-crafted features of the game). T2 enhanced the efficiency of the lookup table approach by integrating an explicit generalization step into the DP backup process. It was tested on Tic-tac-toe. Instead of caching specific Tic-tac-toe boards into a lookup table, T2, relying on an explicit model of its task, applied EBG and goal regression to to produce accurate, large-scale generalizations individual boards. The generalizations were then cached in memory. The application of domain knowledge reduced the size of T2's value function representation from that of a full lookup table and, more importantly, accelerated the DP learning process.

Representing values The approaches described in Sections 6 and 7 draw upon aspects of T2's implementation of the value function. Figure 3 illustrates the type of structures used in T2 to determine values for states. Each possible state value is represented by a tree whose nodes are Boolean features representing subsets of states. The value of a state is determined by comparing it to the features of the trees. When a match is found, the state is deemed to have the value represented by the tree, i.e., to be in the tree's *value class*. The matching of states to value classes requires recursively testing both positive and negative features (depicted as black and white nodes respectively) beginning from the roots of the value class trees. A match occurs when a positive feature is satisfied and none of its negative children are recursively matched. Negative features represent exceptions to their positive parent's "rule." The structure of the value trees thus correspond to *multilevel counterfactuals* (Vere, 1980), in which alternating tree levels have features describing either positive or negative (exceptional) instances of the value class.

The first proposed extension of T2 generalizes the idea of value classes. Instead of defining a value class for a single value, a class may represent a range of values. The range of a value class can be broad or narrow, and it can overlap the ranges of other classes. Thus, a hierarchy of value classes can be defined. One expected advantage of this approach over T2's implementation of value classes is that the learning of the class definitions will be more efficient because the definition of one class is distributed, partially overlapping with many other classes. A second advantage should be the creation of abstract features for representing states which should be induced by the definitions of the abstract value classes, i.e., the classes representing large ranges of values. Overall, using abstract value classes should lead to better scaling properties than T2's single-value classes. In particular, they should support the learning of continuous values much better than T2's discrete representation. This proposal for representing values is discussed more thoroughly in Section 6.

The second proposed extension of T2 generalizes the class definitions which, in T2, were essentially lookup tables. Value class definitions are Boolean functions defined over a state space. Section 7 describes a new approach for approximating Boolean functions as hierarchical connectionist networks. Although it is based on a learning rule developed independently of the T2 work, the overall approximation approach will be specifically designed for compatibility with DP-learning tasks as previously outlined. That is, the approximation approach will try to exhibit the properties deemed necessary and desirable for function approximation in the control and learning framework. Additionally, the approach for approximating Boolean functions will likely incorporate the counterfactual structures used in T2's value class definitions. Work with the T2 system suggests that the counterfactual structures support good generalization and are efficient to evaluate during recognition.

Thus, they are promising structures for defining abstraction hierarchies.

The primary goal of the T2 work was to develop a general method for exploiting task-specific knowledge to enhance the efficiency of DP-based learning methods. Although the current emphasis is on weak approaches, it is hoped that the abstract features formed using the proposed approaches will provide a useful basis for the higher-level, explanation-based reasoning processes used in T2, as well as other methods for exploiting knowledge. Synthesizing strong and weak approaches to generalization should be a promising line for future research, beyond the current proposals.

6 Top-down Abstraction: Value Hierarchies

Dynamic programming exploits the Markov property of system states to optimize backwards, working from the end of a task toward its beginning. The idea that states allow one to ignore the past is both subtle and powerful. It is common to view dynamical systems as existing "out there" in the world, intrinsically characterized by their states and state transitions. In truth, what determines states, and therefore systems, depends on what information an agent wants to predict and control. States are simply the information of primary interest and any secondary information necessary for predicting and controlling the first. Logically, goals precede systems.

Given this observation, determining values for states seems somewhat backward. A more profound issue might be: given goals for controlling certain types of feedback, how should an agent define states? This section describes an approach for learning optimal value functions which is based on the view that one should use values to determine abstract state representations.

Value hierarchies I propose using a hierarchy of *abstract value classes* to induce abstractions over the state space of a task. An abstract value class can be an arbitrary set of values in general, but we restrict attention here to classes representing intervals of integers or reals. Each class represents a single, fixed interval which may differ from other classes in size and may overlap some of them. Value classes are defined over the given state space of a task: a class contains every state x whose optimal value $V^*(x)$ lies within

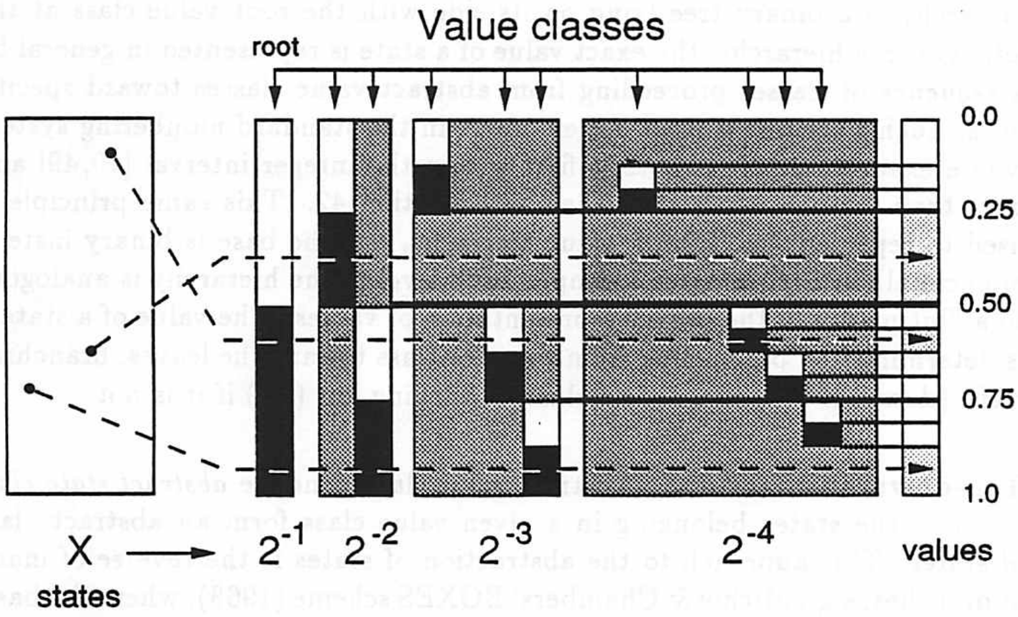

Figure 4: A hierarchy of abstract value classes representing the interval [0.0, 1.0]. The hierarchy is a binary tree lying on its side. Each tree node is an abstract value class representing the interval depicted in black. White intervals denote the states that a class must correctly exclude from its interval. Grey intervals denote *don't care's*, i.e., states for which the class is not responsible. *Don't care's* may be classified in any manner. The value of a state is determined through a sequence of classes proceeding from the root to a leaf.

the class interval. The value classes of T2 are simply abstract value classes representing single-integer "intervals."

Because value classes may overlap, the subset relation partially orders them, forming a hierarchy from the most inclusive, i.e., abstract, classes to the most exclusive or specific classes. For simplicity in the following discussion, we present hierarchies of value classes, *value hierarchies*, structured as binary trees. In the proposed research it is possible that more general hierarchical structures will also be investigated, e.g., lattices.

A single hierarchy represents the values of all states in a space X which are assumed to lie within the global interval spanned by the hierarchy. Figure 4 shows a value hierarchy representing the global real interval [0.0, 1.0]. This hierarchy is a binary tree lying on its side with the root value class at the left. Given a hierarchy, the exact value of a state is represented in general by a *sequence* of classes proceeding from abstract value classes toward specific ones. Such a scheme is used, for example, in the standard numbering system where exact value of, say, 42 is first within the integer interval [40, 49] and only then further resolved to the third position 42. This same principle is used to represent values in a value hierarchy, but the base is binary instead of decimal. In Figure 4, for example, each level of the hierarchy is analogous to a "bit place" in the binary representation of values. The value of a state x is determined by proceeding from the root class toward the leaves, branching right (down) if x is in a node's class, branching left (up) if it is not.

Top-down abstraction Abstract value classes induce ***abstract state classes***, i.e., the states belonging in a given value class form an abstract class of states. This approach to the abstraction of states is the reverse of many approaches, e.g., Michie & Chambers' BOXES scheme (1968), where the basis for abstracting states is a similarity metric defined *a priori* over the input state space. Such approaches operate *bottom-up*, learning outputs for coarse-coded inputs. The values learned for coarse-coded states will approximate the true values of true states *if states were originally abstracted appropriately.* Success, again, depends on either a strong similarity metric or an appropriate input representation.

The value hierarchy approach works *top-down*, defining an *a priori* coarse coding of output values. Learning then defines state abstractions corresponding to the abstract value classes. Top-down abstraction, proceeding from output values to input states, should be a preferable approach in general because it allows the goals and dynamics of a task to determine which regions of the state space to partition coarsely and which to partition finely.

Rationale Learning values through hierarchies is intended to help DP-based methods scale to tasks with large state spaces and/or long delays between actions and their significant payoffs, e.g., the achievement of goal states. The rationale for the approach is that the information in specific value backup can be generalized more effectively when it is in a distributed, hierarchical form. In such a value backup, learning is distributed across several value classes ranging from abstract to specific ones. The more abstract a class is, the more likely it is to participate in many backups. Thus, the most abstract classes will receive a disproportionate share of the training information. This is desirable because they represent the most significant, high-order, bits of states' values—the most important bits to recognize correctly.

Learning the "high-order" value classes should be efficient not only because they receive much training data, but also because no class needs to learn the exact values of any states. Because the most abstract classes represent relatively large ranges of values, one should often be able to exploit very general information to determine accurate classifications for many states. In the current formulation using binary trees, the learning of fine distinctions may still be necessary for some states, especially perhaps for states whose values are close to the boundary of a class interval. Formulations of lattice-structured hierarchies to alleviate this sensitivity have been investigated, and may be used in the research. In any case, the learning of each abstract value class is, by itself, a much less constrained problem than learning the exact values of states, and for the most abstract states this should lead to efficient learning of the value information that is of greatest importance for attaining even moderate levels of performance in a task.

Learning the low-order value classes, which represent the most specific value intervals, is also expected to be efficient. Although they do not generally receive as much training information as the more abstract classes, the more specific classes' learning problems are correspondingly less constrained. A specific class is only responsible for correctly classifying a portion of the states classified by its parent. Ideally in a binary tree structure, each child would receive about one-half of its parent's responsibility. The states for which a class is not responsible, are *don't care's*; they may be classified in any manner. Thus, the more specific a class is, the more freedom it has to exploit *don't care's* in forming generalizations. This should allow more efficient global approximation methods, e.g., linear ones, to succeed in learning the specific classes.

From the standpoint of performance, it is significant that the most quickly learned value classes are expected are the most abstract ones, e.g., those near the root of a tree structure. These classes provide the most important value information for performing reasonably well in a task because they represent the high-order ("low frequency") distinctions between states. It should be possible to propagate such coarse value information much faster and further than the fine distinctions represented by the low-order value classes. Because each value class is only a partial indicator of any state's value, its learning task should be relatively easy. Moreover, generalization is enhanced by the large *don't care* regions in the domains of many classes. When values are represented by a sequence of abstract classes, the complete learning of specific values—i.e., learning to recognize all and only the states with those values—entails the partial learning of many related values, with greater transfer at the more abstract levels of representation. Finally, in addition to generalization it should be possible to propagate abstract value information by using more extensive backups, e.g., by using "differential eligibility traces" (cf. Sutton, 1988), where in performing backups along an action sequence, abstract values are eligible further backward than specific values.

Discussion Forming abstractions top-down, from values to states, is necessary without strong prior knowledge about a given task. Given representations for any two system states for a task, the only basis for judging their similarity are their relationships to the goals of the task, as given by a performance measure for example, and the relative costs of transforming the states in desired ways, which depends on system dynamics. A weak learning system, therefore, can only determine appropriate groupings of states by regressing values backward through the dynamic structure of the controlled system. The efficiency of this process can, of course, be greatly affected by the prior quality of the given input representation of states. Weak approaches cannot depend upon being given good input representations, but when they are available, it would be good to exploit them to make learning more efficient.

Function approximation is required to learn the mappings from states to the classes of a value hierarchy, and a hierarchy itself begins to address the crucial issue of controlling approximation resources. Note that the range of values of V^* for a task might not require representation at a uniform resolution. This is shown in Figure 4 by the varying widths of value intervals across the global interval $[0.0, 1.0]$. In a value hierarchy, it should be possible to allocate more levels of specific classes for critical subranges of values, again, possibly at the cost of reducing resolution for other subranges. An exact algorithm for dynamically allocating value classes will be pursued in the proposed research. Such considerations also raise interesting possibilities for representing and learning the *relative* values of states rather than their absolute values. This question, too, will be studied. Finally, it seems clear that priority in the allocation of function approximation resources generally goes to the more abstract value classes with more specific classes receiving resources to the extent that they are available and that allocating them is justified by improved performance.

7 Address Learning: Units and Networks

Each class in a value hierarchy requires a class-membership definition, i.e., a *characteristic function*, that covers the given states of a task. Figure 5 shows a class definition implemented as a connectionist network. The collection of all such class definitions for a hierarchy constitute a value function for a task. Learning V^* using a value hierarchy, therefore, entails learning class definitions that together form the optimal value mapping from states to the value classes. Any learning approach used should address the function approximation and exploration issues presented in Sections 3 and 4.

This section outlines a novel approach for approximating Boolean functions, which will be used to learn value class definitions. A Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ is implemented as a multilayer feed-forward network composed of *linear threshold units* (LTU's) (e.g., see Nilsson, 1965; Duda & Hart, 1973; Minsky & Papert, 1969). (Nilsson, 1965; Duda & Hart, 1973; Minsky & Papert, 1969) A new rule for training LTU's, called the *address rule*, is presented, and an approach is outlined for constructing hierarchical networks of units that implement the address rule, henceforth called *address units*. Both the address rule and network construction methods are incremental.

Although the approach implements supervised learning from examples, it is specifically designed for the current control and DP-based learning framework. In particular, the approach will facilitate control over generalization behavior through the management of approximation resources, foremost among which are Address units. Address units are LTU's with properties similar to *local basis functions* (Poggio & Girosi, 1989; Kanerva, 1988),

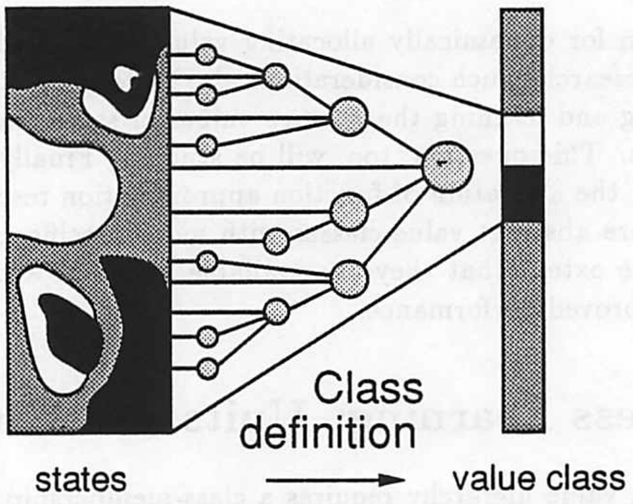

Figure 5: A value class defined by a connectionist network. A characteristic function for each value class is learned by constructing a hierarchical network of linear threshold units.

i.e., they have receptive fields that can be tuned to cover local or global regions of an input space. Hierarchical networks of address units will be used to control resources—units and connections—in a manner analogous to value hierarchies. Hierarchical networks will be composed of address units with large receptive fields at the "top" of the hierarchy and units with very small, local receptive fields at the bottom (cf. Figure 5). The local units are used to correct the errors of the more global units in a manner analogous to the counterfactual (exception-based) structure of value trees used in T2. Thus, as a value hierarchy allows one to place attention and resources on important abstract value classes above the lower-order details, so, too, a hierarchical network allows one to manage, for an *individual* value class, the inevitable tradeoff between approximation accuracy/complexity and the use of resources, e.g., address units. Finally, because the approach has features similar to memory-based instance-based or approaches, it might be possible to provide confidence estimates on values, which could be useful for both efficient exploration and greater robustness in value hierarchy representations.

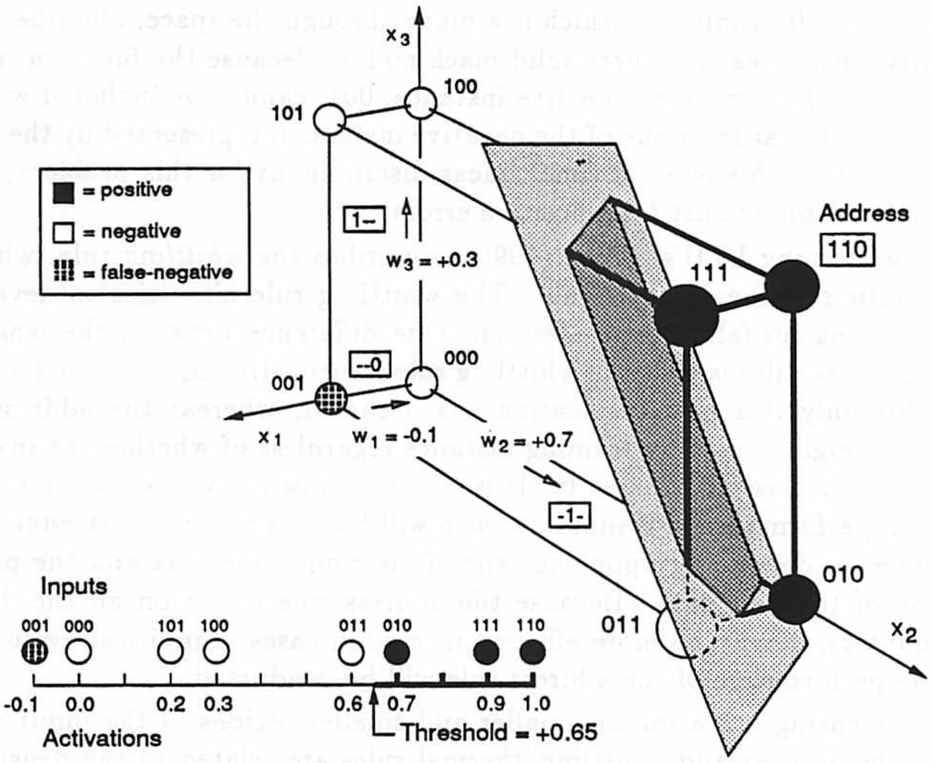

Figure 6: A linear discriminant with no false-positive errors. The instance producing the maximum activation, the address, is determined by the signs of the weights w_i. The weight magnitudes determine distances from the address. The threshold determines the hyper-plane which is set so as to exclude the negative instance nearest the address (011 in the figure).

7.1 The Address rule

The address rule is somewhat unusual among LTU learning rules (cf. the *perceptron* rule) in that its goal is to form an optimal linear discriminant that produces no false-positive errors, i.e., a unit tries to recognize a maximal number of inputs from a particular classes while maintaining 100% accuracy on its positive identifications (i.e., without commission errors). Because many classification tasks are not linearly separable, false-negative (omission) errors must be permitted. Visualizing this goal may help in understanding how the address rule tries to achieve it. Figure 6 depicts an LTU discriminant formed for a non-separable Boolean function on a 3-dimensional input space. The discriminant, which is a plane through the space, identifies three positive instances: the three solid black nodes. Because the function is non-separable, the remaining positive instance, 001, cannot be included without also including at least one of the negative instances, represented by the white nodes. Thus, this is an optimal linear discriminant for this problem, given the restriction against false-positive errors.

For training LTU's, Frean (1990a) describes the *whittling* rule, which is a variation of the *thermal* rule. The whittling rule also tries to develop a unit having no false-positive errors. One difference between the whittling and address rules is that the whittling rule is error-driven, modifying a unit's weights only if a misclassification has occurred, whereas the address rule adapts weights on every training instance regardless of whether the instance is currently classified correctly. It is not yet known how the two rules compare in performance. Presumably, each will have its respective strengths and weaknesses depending upon the type of discrimination task and the presentation of training data. Because the address rule learns on all the data it encounters, it may be more efficient in certain cases. Empirical evaluations of the performance of the address rule will be conducted.

In focusing attention on smaller and smaller regions of the input space, both the address and whittling/thermal rules are related to the density estimation technique of Parzen windows (Duda & Hart, 1973). Other related work includes Frean's (1990b) Upstart algorithm and Gallant's methods for training LTU's and constructing networks from them (Gallant, 1986b; Gallant, 1986a).

The address of a unit An LTU's discriminant is determined by its *weight vector* and its *threshold*. Additionally, Kanerva (1988) describes the *address* of an LTU, a concept which was the inspiration for the address rule. The address of an LTU is the input yielding the maximum possible activation of the unit. The maximum activation is attained by summing all of the unit's positive weights and none of its negative ones. Thus, the address is the input having 1's on all of the positively weighted input dimensions and a 0's on all the negatively weighted ones. Notice that the address is determined solely by the *signs* of the weights; neither the weight magnitudes nor the threshold affect a unit's address.

In Figure 6, for example, the weights are positive, positive, and negative: $x_2 = +0.3$; $x_1 = +0.7$; and $x_0 = -0.1$, where the indices taken left to right are in descending order. Hence, the address is 110. (Determination of the address is illustrated by the small arrows adjacent to the weights: the arrow for each dimension points in the direction of the corresponding address bit.) The number line at the bottom of the figure shows the activations of all inputs. The address input 110 yields the largest activation, 1.0 ($= 0.3 + 0.7$), while its *complement*, 001, yields the smallest activation -0.1. Clearly, the address's complement always produces the smallest activation because its activation is the sum of all the negative weights and none of the positive ones. All other inputs yield activations lying between the activations of the address and its complement.

Activation and distance Inputs lying close to the address in *Hamming distance* [6] generally produce activations greater than those of inputs lying further away. The exact relationship between distances and activations also depends on the magnitudes of weights. The following equalities hold:

$$\begin{aligned} D^*(x) &= \sum_{i=1}^{N} |w_i(x_i^* - x_i)| \\ &= S(x^*) - S(x) \end{aligned}$$

where $D^*(x)$ is the ***absolute weighted Hamming distance*** [7] between an input x and the address x^*; N is the dimension of the input space; w_i and x_i are, respectively, the weight and input on dimension i, and $S(x) = \sum_{i=1}^{N} w_i x_i$ is the activation of input x. Thus, input x yields a greater activation than input y, $S(x) > S(y)$, if and only if x is closer to the address than y, $D^*(x) < D^*(y)$. Note that an input can be close to the address in standard Hamming distance and yet be far in the absolute weighted distance if one or more of the differing dimensions is heavily weighted. Similarly, an input differing from the address on several dimensions may yet be close to the address if all of the differing dimensions have small weights. These observations reveal the function of the

[6] The Hamming distance between two inputs is the number of dimensions with differing bits.

[7] Note that the absolute value $|\cdot|$ in D^*'s summation is unnecessary because of the definition of the address: x_i^* is 1 or 0 exactly when w_i is positive or negative, respectively.

weight magnitudes: they are used to distort distances in the input space, pushing or pulling inputs away from or toward the address. Positive inputs near the address are kept close, while negative inputs are pushed away from the address.

Consider, for example, inputs 100 and 011 in Figure 6. Although 011 is standard Hamming distance 2 from address 110, the two dimensions have small-magnitude weights, −0.1 and 0.3, yielding a total difference of 0.4 from the address. On the other hand, 100 differs from the address only on the second dimension, but the weight on this dimension is 0.7, resulting in a greater difference from the address than 110 and, hence, a correspondingly lower activation. The reason for these differences is that 011 is surrounded by 3 positive inputs positive inputs: 001, 010, and 111. Because both 010 and 111 are within 1 bit of the address they are kept close and, therefore, so is 011. Input 100, on the other hand has two negative neighbors: 000 and 101. Therefore, all of these inputs are kept distant from the address.

The linearity of an LTU strictly limits the flexibility of adjusting distances, via weight magnitudes, to locate negative and positive instances relative to the address. In particular, all inputs lying on a minimal path [8] between the address and its complement will always have activations whose ordering is the same as the ordering of the inputs along the path (Kanerva, 1988). In other words, no combination of weights can give an input a greater activation than another input lying closer (in standard Hamming distance) to the address on along some minimal path. For example, 000 will always have a lower activation than either 100 or 010, given that 110 is the address.

Forming a neighborhood of the address and setting the threshold

Within the limitations of linearity, however, it is possible to adjust the weight magnitudes so as to push negative instances away from the address while keeping neighboring positive instances nearby. The address rule can thus be described qualitatively. Given an LTU that is to identify positive instances of one class with 100% reliability, the rule tries foremost to locate a good address. A good address is a positive (or, possibly, unobserved) in-

[8]A minimal path between two instances is a sequence of instances obtained by flipping once each bit on which the two instances differ, thereby transforming one instance into the other in a minimal number of steps. Different minimal paths correspond to different orderings of the bit selections.

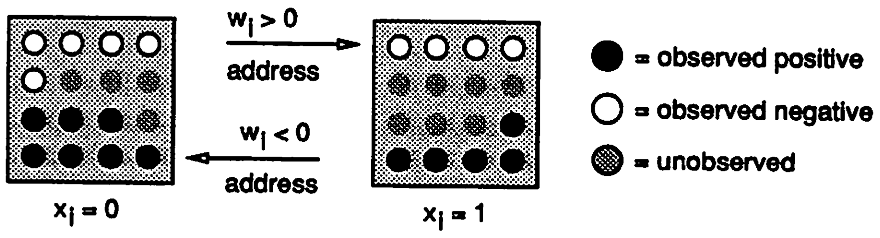

Figure 7: The input space separated on the i^{th} dimension.

stance that is densely surrounded by many other positive instances. The address is determined by the signs of the LTU's weights. Concurrently, the learning rule tries to push negative instances away from the address while keeping, to the extent possible, other positive instances nearby. These relative distances are determined by the weight magnitudes. The aim is to settle upon a configuration in which a large neighborhood of positive instances has been located and isolated at the high-activation end of the range of input activations. The address will have the maximum activation and will roughly correspond to the "center" of the neighborhood.

Having formed such a neighborhood, the LTU's threshold is set higher than the largest activation of any negative instance. This ensures that no false-positive identifications are made. Setting the threshold defines a boundary around the address. The bounded region should exclude all negative instances while including as many positive instances as possible.

Learning the weights Each input dimension i partitions instances into two sets depending upon the value of the instance component x_i. This is illustrated in Figure 7 which also shows that each set is composed of observed positive or negative instances and unobserved ones (black, white, and grey, respectively). The sign of the corresponding weight w_i determines which set will contain the address: if $w_i > 0$, the i^{th} bit of the address is 1; if $w_i < 0$, the i^{th} bit is 0. How should w_i be chosen so as to select the best set for the address?

For the moment, let us ignore the influences of weights on dimensions other than i. An obvious heuristic for choosing the best address set on the i^{th} dimension is to choose the one having the most positive instances and the fewest negative ones. Of course, it is possible that a single set could have the most of both types. In general, then, one must distinguish four types of instances, corresponding to the two classes, positive and negative, and the two values of the i^{th} bit. Let the four parameters $P_i\,[pos,0]$, $P_i\,[pos,1]$, $P_i\,[neg,0]$, and $P_i\,[neg,1]$ represent the number of instances of the corresponding type that have been observed for dimension i. The set $x_i = 1$ ($w_i > 0$) should be a good choice for the address if there are many positive instances whose i^{th} bit is 1 ($P_i\,[pos,1]$ instances), *or* if there are many negative instances whose i^{th} bit is 0 ($P_i\,[neg,0]$ instances, indicating that the set $x_i = 0$ ($w_i < 0$) would be a *bad* choice). Clearly, $x_i = 0$ would be a good choice in the reversed situation: $P_i\,[pos,0]$ and/or $P_i\,[neg,1]$ large.

It is reasonable, therefore, for w_i to have the same sign as the corresponding first-order Walsh coefficient:

$$diff_i = (P_i\,[pos,1] + P_i\,[neg,0]) - (P_i\,[pos,0] + P_i\,[neg,1])\,.$$

Moreover, the *magnitude* of $diff_i$ also provides a reasonable measure for determining the magnitude of w_i. The greater the difference between the two subexpressions in $diff_i$, the less likely it is that the non-address set will contain many positive instances sufficiently close to the address to be included in its neighborhood. On the contrary, it is more likely that the non-address set will contain many negative instances close to the address (in standard Hamming distance). Thus, in such cases, separating the two half-spaces by giving w_i a large magnitude should allow a larger neighborhood to be formed around the address.

Figure 8 illustrates the sign and magnitude of $diff_i$ for three types of data. In case 1, the data are nearly evenly balanced between preferring the address in set $x_i = 0$ and $x_i = 1$. The magnitude of $diff_i$ is small in such a case, and its sign is essentially arbitrary. In case 2, there is a small but distinct trend indicating that $x_i = 0$ is preferred for the address. The magnitude of $diff_i$ is still rather small, but, barring a shifting trend, its sign is not likely to change with future data. In case 3, the data overwhelmingly favor locating the address in set $x_i = 1$. The magnitude of $diff_i$ is large, so its sign is essentially fixed: a large number of significantly different data would be required to change it.

A simple way to determine w_i is to set it equal to the normalized value

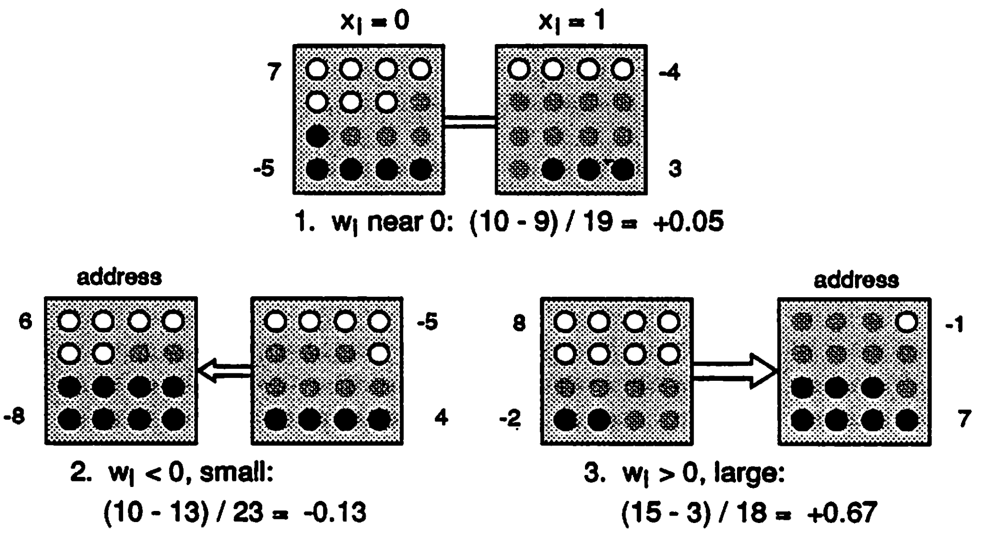

Figure 8: Three examples of how data affect $diff_i$.Case 1: $diff_i$'s magnitude is near zero, and its sign is arbitrary. Case 2: $diff_i$ has a small, negative value which could be changed if the data were to shift. Case 3: $diff_i$ is large and positive; its sign is essentially fixed.

of $diff_i$:

$$w_i = \frac{diff_i}{P_i[pos,1] + P_i[neg,0] + P_i[pos,0] + P_i[neg,1]}. \tag{5}$$

This ensures that as weights are updated, they converge toward fixed values. It also constrains weight values to lie in the range $[-1, +1]$. Convergence of the weights is a desirable property for several reasons which will be discussed below. It is less clear whether bounding the weights yields any significant advantages or disadvantages.

Weighting the instances Unfortunately, learning each weight simply according to rule (5) does not always lead to the identification of a good address. The weakness is that each weight considers only the numbers of positive and negative instances in its two half-spaces. However, the locations of the instances is also important. Because false-positive classification errors are prohibited, a good address must be located in a *dense* region of positive instances. Even sparsely distributed negative instances near the address can severely restrict the size of the neighborhood that can be formed. (Later, we will consider ways of relaxing such stringent conditions, but the goal of locating an address in a reasonably dense positive region will remain.) Thus, rule (5) must be augmented with a mechanism that allows all of the weights to influence each other so that each will be sensitive, not only to the numbers of its various types of instances, but also to their locations, as viewed by weights along the other dimensions.

To illustrate both the problem and a simple solution, consider the 4-dimensional space shown in Figure 9. If each weight were to compute its value from (5) independently, then node 1010 would be identified as the address. Notice that the third dimension selects 0 as its address bit: there are slightly more positive instances in the bottom half-space (five with $x_3 = 0$) than in the top (four with $x_3 = 1$).

Unfortunately, 1010 is not the best address. A neighborhood of only four positive instances can be formed around it. Positive instances 0111 and 1111 cannot be included because negative instances 0011 and 1011, respectively, lie on a minimal paths between them and 1010. A better address would result by flipping the third dimension's address bit to 1, making node 1110 the address. One can form a neighborhood around 1110 that includes instances 0111 and 1111 in addition to the previous four instances, giving a total of six.

How can one cause the weight on the third dimension to select 1 rather than 0? Notice that the most significant cluster of positive instances is in the right-hand half-space (i.e., $x_2 = 1$). This is reflected in the fact that, for weight w_2, diff_2 has a magnitude of 6 while the other three diff_i have magnitudes of only 2. Notice further that in the right-hand half-space, all four of the upper instances (0110, 0111, 1110, and 1111) are positive, while only two of the lower ones (0010 and 1010) are positive. Thus, in this half-space, $x_3 = 1$ is clearly preferred over $x_3 = 0$. If the instances in this half-space ($x_2 = 1$) carry great enough influence over w_3—*vis-à-vis* those in the left half-space ($x_2 = 0$)—then w_3 will select an address bit of 1 rather than 0. This interaction between the weights is represented in Figure 10 which illustrates the difference in w_3's view of instances before and after factoring in the influence from w_2.

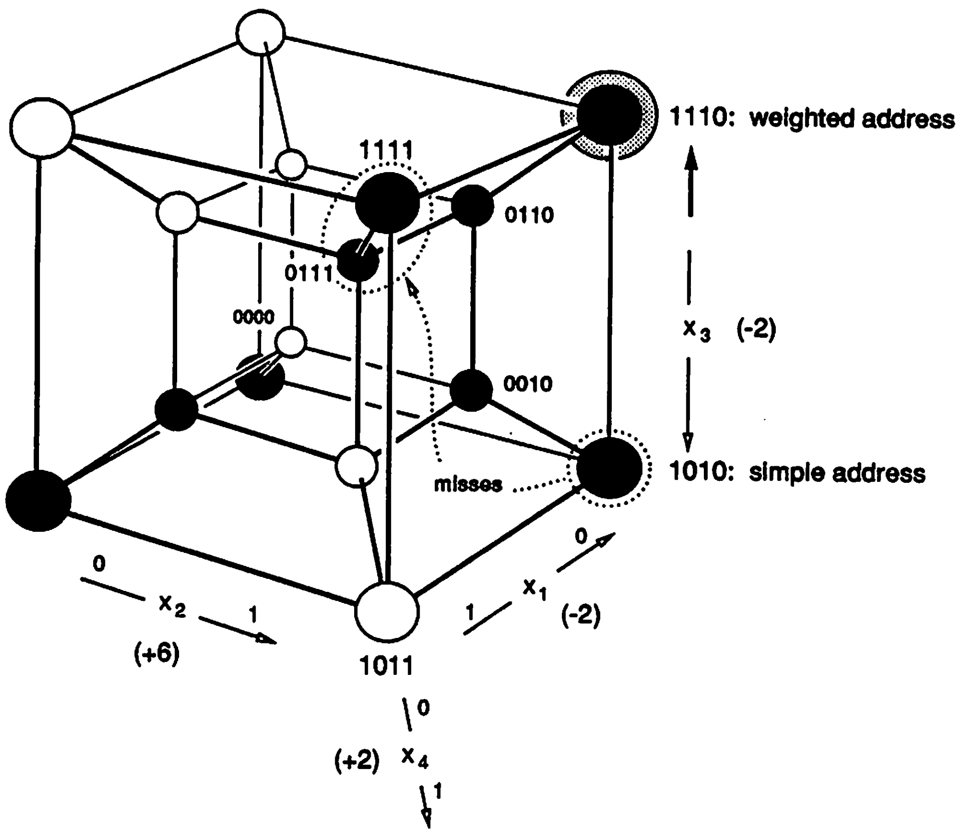

Figure 9: Interdependencies among weights on a 4-dimensional problem.

A simple method for determining the relative influence of instances is to weight them according to their weighted Hamming distances from the current estimate of the address. Instances further from the estimated address, i.e., ones with lower activations, will be diminished in their ability to influence weight updating relative to instances closer to the estimated address. Figure 11 illustrates qualitatively how instances differentially influence weight updating. Each positive instance tries to move the weights so as to reduce the distance between itself and the address. In effect, each positive instance tries to move the address to itself. Conversely, each negative instance tries to move the address away from itself, to its complement. The relative degree to which an instance can move the address is determined by an *instance weight* which is made proportional to the instance's distance from the current ad-

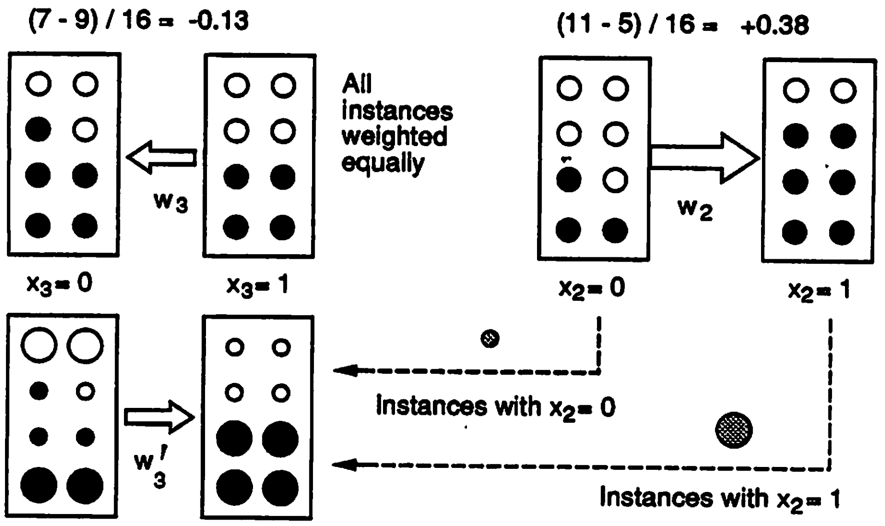

Figure 10: The influence of one weight upon another. For the space shown in Figure 9, weight w_2 strongly prefers an address with $x_2 = 1$ while, initially, w_3 slightly prefers one with $x_3 = 0$. Weighting instances according to w_2, however, causes w_3 to change its preference to $x_3 = 1$, which yields a better address.

dress. Perhaps the most straight-forward weight to use is the ratio between the instance activation and the maximum possible activation S^{max} (i.e., the address's activation), using the minimum possible activation S^{min} (the complement's activation) as the baseline, rather than 0. Thus, the weight of instance x is:

$$inst\text{-}wt = \frac{S(x) - S^{min}}{S^{max} - S^{min}}. \tag{6}$$

Differential convergence Weights do not select their values independently. They are coupled through the feedback mechanism of instance weighting. Large weights have more influence than small ones because they have a greater capacity for diminishing the influence of instances that do not agree with the weight's choice of address bit. By thus distinguishing among instances, the large weights provide a context. A large magnitude on a weight implies a significant asymmetry in the distribution of positive and negative instances between the two half-spaces of the corresponding dimension. A

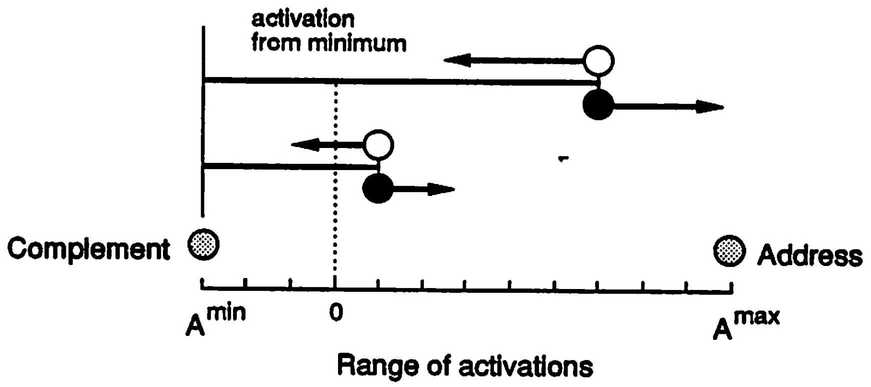

Figure 11: Weighting instances by distance from the address. The ability of instances to influence the adjustment of weights diminishes with increasing distance from the current address.

set of large weights identifies a subspace of the input space that is likely to contain a good address. Smaller weights may then further refine the selected subspace along dimensions that previously held no strong preference for the location of the address.

In order for the smaller weights to select a good address—perhaps reversing their initial tendencies as in the case of w_3 in the example of Figure 9—they must remain adaptable until the more significant weights clearly identify a good subspace for the address. Because weights converge toward fixed values, smaller weights should converge more slowly than larger ones. This is currently implemented by controlling the frequency with which weights update. Each training instance is weighted according to (6). For each weight w_i, the instance weight is added to the appropriate parameter, $P_i\,[pos, 1]$, $P_i\,[pos, 0]$, $P_i\,[neg, 1]$, or $P_i\,[neg, 0]$. If either the sum $(P_i\,[pos, 1] + P_i\,[neg, 0])$ or the sum $(P_i\,[pos, 0] + P_i\,[neg, 1])$ exceeds 1.0 (any reasonably small constant should do as well, e.g., 2.0 or 10.0), then the weight is updated according to (5). This means that weights for which there is a strong asymmetry in the types of instances encountered will tend to update more frequently (hence converge sooner) than weights for which the instances are more balanced. These more frequently updating weights will also be the larger ones because the magnitude of the difference $diff_i$ will be larger than for the more balanced weights.

Input: An LTU, an input instance x, and its classification *x-class*.

Output: The weights of the LTU have been updated so as to better identify a good address.

Method: All weights w_i and their corresponding 4 parameters $P_i[\cdot,\cdot]$ are initialized to 0. K is a constant controlling the relative frequency at which a weight can be updated. A typical value of K would be 1.0 or 2.0 but might range up to 10.0. The *Rec-mean$_i$* are recursively updated averages. (Note: the LTU's threshold is updated in a separate procedure.)

```
$S^{max} \leftarrow$ sum of positive weights
$S^{min} \leftarrow$ sum of negative weights
If $(S^{max} = S^{min})$ then  $x\text{-}wt \leftarrow 1$
                          else  $x\text{-}wt \leftarrow (A(x) - S^{min}) \,/\, (S^{max} - S^{min})$
For each weight $w_i$:
  1. $P_i\,[x\text{-}class, x_i] \leftarrow P_i\,[x\text{-}class, x_i] + x\text{-}wt$
  2. If $(P_i\,[pos,1] + P_i\,[neg,0] > K)$ or $(P_i\,[pos,0] + P_i\,[neg,1] > K)$ then
     (a) $w_i \leftarrow Rec\text{-}mean_i\,(P_i\,[pos,1] + P_i\,[neg,0] - P_i\,[pos,0] - P_i\,[neg,1])$
     (b) Reset all four parameters $P_i[\cdot,\cdot]$ back to 0.
```

Table 3: The algorithm for updating weights.

As the larger weights converge forming a subspace for the address, instances within the subspace will consistently outweigh their counterparts outside the subspace. This introduces a bias that may cause previously balanced weights to strongly favor instances from one of their half-spaces over those from the other. This, in turn, introduces new biases that may further the convergence of still more weights. Thus, there is a cascading effect in the convergence of the weights of an Address unit.

Discussion Table 7.1 describes the weight updating algorithm in pseudocode. This version of the algorithm is still preliminary. It has been tested on several problems and works reasonably well, but further testing or theoretical investigations might uncover deficiencies or suggest enhancements. Because weights are computed in a manner suggestive of Walsh coefficients, it might be possible to develop a more coherent mathematical framework for the rule, one relating it to more traditional digital signal processing techniques. Weight values also seem closely related to the correlation between the value of the input bit x_i and the instance class, positive or negative. However, the relationship between these two measures is also not yet clear to me.

Another unsettled issue is the means for propagating constraints between weights so that weights having clearly asymmetric distributions will influence ones having more balanced distributions. The current version of the algorithm discounts instances by the same amount for all weights but weight-updating may occur at varying rates. The constant K affects the rate of updating, larger values causing a greater difference between fast and slow weights. One possibility might be to "anneal" K, starting it at a large value so that there is initially a large difference between weights, and moving it to a small value as the larger weights converge. Another alternative might be to allow weights to update at the same rate but discount instances differently for different weights. It is not clear whether such an approach would be a more effective way of locating an address in a dense region of positive instances. In short, the current algorithm could still undergo substantial revision in its mechanisms or even, possibly, in the heuristic principles underlying them.

Learning the threshold Unlike many connectionist learning methods which treat the threshold or bias as a special weight to an input of constant value 1, the threshold θ of an Address unit is adjusted separately from the weights. The purpose of the threshold is to define a homogeneous neighborhood of positive instances around the address. Thus, a straight-forward setting for θ would be slightly above the largest activation of any negative instance. This would produce largest possible neighborhood. However, if there are unobserved instances in such a neighborhood, it is possible that some of them could prove to be negative, in which case θ would have to be readjusted upward to exclude them. Call the observed negative instance with the largest activation *max-neg*. A more conservative setting for θ would be slightly below the lowest activation of any positive instance whose activation is above that of *max-neg*. Call such a positive instance *min-pos*.

In general, as the weights are adjusted, the activations of both *max-neg* and *min-pos* will fluctuate, especially during the early stages of learning. In fact, the actual instances filling these roles will change. Thus, the current version of the algorithm keeps a running record of *max-neg* and *min-pos*, and dynamically computes the threshold θ to be the average of their two activations:

$$\theta = \frac{S\,(\textit{min-pos}) + S\,(\textit{max-neg})}{2.0}. \tag{7}$$

Since the classifications produced are likely to be unreliable until a reasonably good address has been located, an even more conservative approach would start θ at a value very close to that of S^{max} and gradually lower it to a value such as that in (7).

7.2 Networks of Address units

Because a single address unit can only implement linearly separable Boolean functions, it is necessary to consider more general approximation methods. This section sketches an approach to constructing networks of address units which can implement broader classes of Boolean functions. The networks must be designed to learn in a manner well-suited to the DP framework.

The construction of address networks will take advantage of the special properties of address units. Because they are trained to recognize locally dense regions of positive instances, address units act as local basis functions (Poggio & Girosi, 1989; Kanerva, 1988). A unit's receptive field is defined by its address, which acts as a "center," and its threshold, which defines a "radius." The relative size of a unit's receptive field is determined by the relative amount of the unit's range of weights, from S^{max} to S^{min}, that lies above the threshold θ. If θ is close to S^{max}, then the unit will respond to inputs only from a very small neighborhood of the address. If θ is close to S^{min}, then the unit will respond to nearly the entire input space.

A unit's address can be set to a particular input and its threshold θ can be kept very close to S^{max}. Such a unit thus effectively memorizes the classification of a specific positive input. Additional training can be used for *local* generalization by maintaining constraints on θ. Such units thus act like cells in a lookup table that perform local generalization, much like a CMAC (Albus, 1975b). Units whose thresholds are a significant percentage of their weight range $[S^{min}, S^{max}]$ or who have a number of weights close to zero, have large receptive fields. Rather than memorizing specific positive inputs, such units act more like general "rules" for classifying positive inputs of a certain type. By constraining a unit's threshold and weights, therefore, one can control whether a unit acts to memorize specific positive data or tries to identify abstract subclasses of such data. Allocating such different types of address units will be the primary mechanism for constructing hierarchical networks.

The larger a unit's receptive field is, the more likely it is to include negative instances, unless the target function happens to be nearly separable. Such negative instances can be considered as exceptions to rules. Thus, to prevent false-positive errors for such "abstract" units, more specific units, with smaller receptive fields, will be used to recognize negative instances (exceptions) and prevent more abstract units from responding to them, resulting in a hierarchical counterfactual structure similar to the value trees used in T2. Such an approach is also related to Salzberg's (1991) function approximation method which employs structures with nested exceptions.

The construction of a hierarchical network structure will proceed from the specific to the abstract. That is, specific units will first be to used to memorize and classify incoming data. More abstract units will then learn, probably more slowly, to perform the work of two or more specific units of the same class and when sufficiently accurate, will supplant the more specific units. Because large receptive fields may include smaller areas of exceptions, abstract units may recruit more specific units of the opposite class to override potential classification errors. This enables abstract units to build up very large receptive fields and still respect the prohibition against false-positive errors.

The replacement of many specific units by fewer, more-abstract ones is expected to lead to better overall generalization because the "rules-plus-exceptions" structure should to produce concise and efficient implementations for many function approximation problems encountered in practice. Also, if resources must be reclaimed, then the more specific units will most likely be deleted first. Thus, two hierarchical structures will be used to control the allocation of resources: value hierarchies tend to concentrate resources on more abstract value classes, and within the definition of each class, hierarchical address networks concentrate resources on abstract units which represent general subclasses of states. Ultimately, the goal is to characterize a value function using a small number very abstract value classes and address units, with more specific classes and units used as needed to refine performance.

8 Discussion

The central goal of this research is to enhance the scaling properties of DP-based learning methods, extending their applicability to tasks of greater size and complexity. Toward this end, we have examined function approximation for learning a task's optimal value function V^*, focusing on the issue of generalization and representational bias. Because little *a priori* structure is imposed on Markov decision tasks, DP-based learning calls for approximation methods that initially produce only local generalizations. However, learning efficiency requires that as experience accrues, generalization behavior accelerates toward strong, task-specific biases. Therefore, I have advocated the acquisition of representational biases through the formation of hierarchies of abstract values and states. The mechanism for strengthening generalization biases is the increasing restriction of approximation resources, which forces initially rote memories to abstract the information they contain.

Using hierarchies to allocate resources allows detailed information about values and states to become abstracted in accordance with task goals and constraints imposed by system dynamics. Value hierarchies induce abstract classes of states in a top-down fashion, reducing the need for highly engineered state representations that encode task-specific biases. State classes can be defined using address units, which form receptive fields defining positive or negative sets of class members. A unit's receptive field can be kept local or allowed to cover a large region of the state space. This facilitates the construction of hierarchical networks and, consequently, the control of approximation resources.

Because both value hierarchies and address learning are based on heuristics, I do not anticipate deriving significant theoretical results. The approaches will thus be tested empirically to determine their appropriateness for the value iteration process of DP. Testing can include specially designed function approximation problems in addition to control tasks. Initial testing should establish the ability of the approaches to handle the special function approximation concerns of the control and DP framework. Subsequent testing should then focus on the generalization and scaling properties of the approaches, determining how task performance and resource usage (both time and memory) scale with task size and complexity. The hierarchical control of approximation resources is expected to provide reasonable leverage in confronting the difficulties of function approximation in DP-based learning methods.

In concentrating on values and states, many important issues in optimal control have not been directly addressed. These include the formation of Markov state descriptions, abstract representations of policies and actions, the learning of various types of models, interactions between value functions and policies, planning in abstraction spaces, and principled exploitation of prior knowledge of tasks. In some cases it might be possible to combine current proposals with existing approaches to such problems or, at least, to synthesize new approaches in reasonably straightforward ways. In other cases it might be necessary to rethink the boundaries that have been drawn between these issues. In any event, it is clear that the issue of representation will remain central. The proposed approaches for developing meaningful abstractions are perhaps step toward providing learning agents with somewhat greater autonomy.

Acknowledgements

I am grateful to Andrew Barto, my advisor, who has shaped this research at many levels. I thank Satinder Singh, Vijaykumar Gullapalli, and Steven Bradtke for valuable discussions on issues and approaches related to this research. Paul Smolensky, Paul Utgoff, and Sharad Saxena helped to focus my attention on representational issues in learning. Rich Sutton helped to make the view of bias acquisition more explicit. In addition to Andy Barto, helpful comments on a draft of this report were provided by S. Bradtke, Charles Clifton, and S. Singh.

References

Albus, J. S. (1975a). Data storage in the cerebellar model articulation controller. *Journal of Dynamic Systems, Measurement, and Control*, *97*, 228–233.

Albus, J. S. (1975b). A new approach to manipulator control: the cerebellar model articulation controller. *Journal of Dynamic Systems, Measurement, and Control*, *97*, 220–227.

Alpaydin, E. (1991). GAL: Networks that grow when they learrn and shrink when they forget. Technical Report TR-91-032, International Computer Science Institute.

Atkeson, C. G. & Reinkensmeyer, D. J. (1990). Using associative content-addressable memories to control robots. In Miller, W. T., Sutton, R. S., & Werbos, P. J. (Eds.), *Neural Networks for Control*, chapter 11, pages 255–285. Cambridge, MA: Bradford Books/MIT Press.

Bachrach, J. R. (1991). A connectionist learning control architecture for navigation. In Lippmann, R. P., Moody, J. E., & Touretzky, D. S. (Eds.), *Advances in Neural Information Processing Systems 3*, pages 457–463, San Mateo, CA. IEEE, Morgan Kaufmann.

Barto, A. G. (1990). Connectionist learning for control: An overview. In Miller, W. T., Sutton, R. S., & Werbos, P. J. (Eds.), *Neural Networks for Control*, chapter 1, pages 5–58. Cambridge, MA: Bradford Books/MIT Press.

Barto, A. G. (1991). Some learning tasks from a control perspective. In Nadel, L. & Stein, D. L. (Eds.), *1990 Lectures in Complex Systems*, pages 195–223. Redwood City, CA: Addison-Wesley. Advanced Book Program.

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1991). Real-time learning and control using asynchronous dynamic programming. Technical Report TR-91-57, Department of Computer Science, University of Massachusetts, Amherst.

Barto, A. G. & Singh, S. P. (1991). On the computational economics of reinforcement learning. In Touretzky, D. S., Elman, J. L., Sejnowski, T. J., & Hinton, G. E. (Eds.), *Connectionist models: Proceedings of the 1990 Summer School*, pages 35–44, San Mateo, CA. Morgan Kaufmann.

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, *13*(5), 834–846.

Barto, A. G., Sutton, R. S., & Watkins, C. J. C. H. (1990). Learning and sequential decision making. In Gabriel, M. & Moore, J. (Eds.), *Learning and Computational Neuroscience: Foundations of Adaptive Networks*, chapter 13, pages 539–602. Cambridge, MA: Bradford Books/MIT Press.

Bellman, R. (1957). *Dynamic Programming.* Princeton, NJ: Princeton University Press.

Bellman, R. (1971). *Introduction to the Mathematical Theory of Control Processes*, volume 2. New York: Academic Press.

Bellman, R. E. & Dreyfus, S. E. (1962). *Applided Dynamic Programming.* Princeton, NJ: Princeton University Press.

Benjamin, P., Holte, R., & Rendell, L. (Eds.). (1988). *Proceedings of the First International Workshop in Change of Representation and Inductive Bias*, Briarcliff Manor, NY 10510. Philips Laboratories, North American Philips Corporation.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the Association for Computing Machinery*, *18*(9), 509–517.

Berry, D. A. & Fristedt, B. (1985). *Bandit Problems: Sequential Allocation of Experiments.* London: Chapman and Hall.

Bertsekas, D. P. (1987). *Dynamic programming: Deterministic and stochastic models.* Englewood Cliffs, NJ: Prentice Hall.

Bertsekas, D. P. & Tsitsiklis, J. N. (1989). *Parallel and Distributed Computation: Numerical Methods.* Englewood Cliffs, NJ: Prentice Hall.

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1987). Occam's razor. *Information Processing Letters, 24*, 377–380. Reprinted in J. W. Shavlik and T. G. Dietterich (Eds.), (1990), *Readings in Machine Learning*, Morgan Kaufmann Publishers:San Mateo, CA.

Dayan, P. (1991). Navigating through temporal difference. In Lippmann, R. P., Moody, J. E., & Touretzky, D. S. (Eds.), *Advances in Neural Information Processing Systems 3*, pages 464–470, San Mateo, CA. IEEE, Morgan Kaufmann.

Dean, T. & Wellman, M. (1991). *Planning and Control.* San Mateo, CA: Morgan Kaufmann.

DeJong, G. & Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning, 1*(2), 145–176.

Dietterich, T. G. (1982). Learning and inductive inference. In Cohen, P. R. & Feigenbaum, E. A. (Eds.), *The Handbook of Artificial Intelligence*, chapter XIV, pages 323–511. Redwood City, CA: Addison-Wesley.

Duda, R. O. & Hart, P. E. (1973). *Pattern Classification and Scene Analysis.* A Wiley-Interscience Publication. New York: Wiley and Sons.

Frean, M. (1990a). *Small Nets and Short Paths: Optimising Neural Computation.* PhD thesis, University of Edinburgh, Department of Physics, Edinburgh EH9 3JZ, UK.

Frean, M. (1990b). The Upstart algorithm: A method for constructing and training feedforward neural networks. *Neural Computation, 2*(2), 198–209.

Gallant, S. I. (1986a). Optimal linear discriminants. In *IEEE Proceedings of the Eighth Conference on Pattern Recognition*, pages 849–852.

Gallant, S. I. (1986b). Three constructive algorithms for network learning. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 652–660.

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

Hinton, G. E. (1989). Connectionist learning procedures. *Artificial Intelligence, 40*(1–3), 185–234. Special volume on machine learning, J. G. Carbonell (Editor).

Hinton, G. E. (1990). Preface to the special issue on connectionist symbol processing. *Artificial Intelligence, 46*(1–2), 1–4.

Howard, R. A. (1960). *Dynamic Programming and Markov Processes.* Cambridge, MA: MIT Press.

Jacobs, R. A. (1990). *Task Decomposition Through Competition in a Modular Connectionist Architecture.* PhD thesis, Department of Computer and Information Science, University of Massachusetts, Amherst.

Jordan, M. I. & Jacobs, R. A. (1990). Learning to control an unstable system with forward modeling. In Touretzky, D. S. (Ed.), *Advances in Neural Information Processing Systems 2*, pages 324–331, San Mateo, CA. IEEE, Morgan Kaufmann.

Jordan, M. I. & Rumelhart, D. E. (1990). Forward models: Supervised learning with a distal teacher. Occasional paper 40, MIT, Cambridge, MA.

Kaelbling, L. P. (1990). *Learning in embedded systems.* PhD thesis, Department of Computer Science, Stanford University, Teleos Research, Palo Alto, CA. Technical report: TR-90-04.

Kanerva, P. (1988). *Sparse Distributed Memory.* Cambridge, MA: Bradford Books/MIT Press.

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning, 1*(1), 11–46.

Lane, S. H., Handelman, D. A., & Gelfand, J. J. (1991). Higher-order CMAC neural networks – Theory and practice. Department of Psychology, Princeton University. Presented at the 1991 American Control Conference, Boston, MA.

Le Cun, Y., Denker, J. S., & Solla, S. A. (1990). Optimal brain damage. In Touretzky, D. S. (Ed.), *Advances in Neural Information Processing Systems 2*, pages 598–605, San Mateo, CA. IEEE, Morgan Kaufmann.

Matheus, C. J. (1991). The need for constructive induction. In Birnbaum, L. A. & Collins, G. C. (Eds.), *Machine Learning: Proceedings of the Eighth International Workshop (ML91)*, pages 173–177, San Mateo, CA. Morgan Kaufmann.

Michie, D. & Chambers, R. A. (1968). BOXES: An experiment in adaptive control. In Dale, E. & Michie, D. (Eds.), *Machine Intelligence 2*, pages 137–152. Oliver and Boyd.

Miller, III, T. W., Sutton, R. S., & Werbos, P. J. (Eds.). (1990). *Neural Networks for Control.* Neural Network Modeling and Connectionism. Cambridge, MA: Bradford Books/MIT Press.

Minsky, M. & Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry.* Cambridge, MA: MIT Press. Reissued in an expanded edition, 1988.

Minton, S., Carbonell, J. G., Knoblock, C. A., Kuokka, D. R., Etzioni, O., & Gil, Y. (1989). Explanation-based learning: A problem solving perspective. *Artificial Intelligence, 40*(1–3), 63–118. Special volume on machine learning, J. G. Carbonell (editor).

Mitchell, T. M. (1980). The need for biases in learning generalizations. Technical Report CBM-TR-117, Department of Computer Science, Rutgers University. Reprinted in J. W. Shavlik and T. G. Dietterich (Eds.), (1990), *Readings in Machine Learning*, Morgan Kaufmann Publishers:San Mateo, CA.

Mitchell, T. M., Keller, R., & Kedar-Cabelli, S. (1986). Explanation-based generalization: A unifying view. *Machine Learning, 1*(1), 47–80.

Moody, J. (1989). Fast learning in multi-resolution hierarchies. In Touretzky, D. S. (Ed.), *Advances in Neural Information Processing Systems 1*, pages 29–39, San Mateo, CA. IEEE, Morgan Kaufmann.

Moore, A. W. (1990). *Efficient Memory-based Learning for Robot Control.* PhD thesis, Trinity Hall, University of Cambridge, Cambridge, England.

Moore, A. W. (1991). Knowledge of knowledge and intelligent experimentation for learning control. AI Lab, MIT.

Mozer, M. C. & Bachrach, J. R. (1991). SLUG: A connectionist architecture for inferring the structure of finite-state environments. *Machine Learning*, 7(2/3), 139–160.

Mozer, M. C. & Smolensky, P. (1989). Skeletonization: A technique for trimming the fat from a network via relevance assessment. In Touretzky, D. S. (Ed.), *Advances in Neural Information Processing Systems 1*, pages 107–115, San Mateo, CA. IEEE, Morgan Kaufmann.

Nilsson, N. J. (1965). *Learning Machines: Foundations of Trainable Pattern-Classification Machines.* New York: McGraw-Hill.

Nowlan, S. & Hinton, G. E. (To appear). Simplifying neural networks by soft weight-sharing. In *Advances in Neural Information Processing Systems 4*, San Mateo, CA. IEEE, Morgan Kaufmann.

Omohundro, S. M. (1987). Efficient algorithms with neural network behavior. *Complex Systems, 1*, 273–347.

Peng, J. & Williams, R. J. (Submitted). Efficient search control in Dyna. College of Computer Science, Northeastern University. Submitted to the 1992 Machine Learning Conference.

Poggio, T. & Girosi, F. (1989). A theory of networks for approximation and learning. A.I. Memo No. 1140, Artificial Intelligence Laboratory, MIT.

Poggio, T. & Girosi, F. (1990). Networks for approximation and learning. *Proceedings of the IEEE, 78*(9), 1481–1497.

Rendell, L. (1985). Substantial constructive induction using layered information compression: Tractable feature formation in search. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 650–658.

Rivest, R. L. & Schapire, R. E. (1987). A new approach to unsupervised learning in deterministic environments. In Langley, P. (Ed.), *Proceedings of the Fourth International Workshop on Machine Learning*, pages 364–375.

Ross, S. (1983). *Introduction to Stochastic Dynamic Programming.* New York: Academic Press.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Volume 1: Foundations*, chapter 8, pages 318–362. Cambridge, MA: Bradford Books/MIT Press.

Salzberg, S. (1991). A nearest hyperrectangle learning method. *Machine Learning, 6*(3), 251–276.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development, 3*, 210–229. Reprinted in E. A. Feigenbaum and J. Feldman, (Eds.), *Computers and Thought*, McGraw-Hill, New York, 1963.

Samuel, A. L. (1967). Some studies in machine learning using the game of checkers. II—recent progress. *IBM Journal on Research and Development, 11*, 601–617.

Saxena, S. (1991a). An algorithm to evaluate instance representations. Technical Report 91-21, Department of Computer and Information Science, University of Massachusetts, Amherst, MA 01003.

Saxena, S. (1991b). *Predicting the effect of instance representation on inductive learning.* PhD thesis, University of Massachusetts, Amherst, Department of Computer and Information Science, University of Massachusetts, Amherst, MA 01003. COINS Technical Report 91-58.

Schapire, R. E. (1988). Diversity-based inference of finite automata. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA.

Schlimmer, J. C. & Granger, R. H. (1986). Incremental learning from noisy data. *Machine Learning, 1*, 317–354.

Schmidhuber, J. (1991a). Adaptive confidence and adaptive curiosity. Technical Report FKI-149-91, Institut für Informatik, Technische Universität München, Arcisstr. 21, 800 München 2, Germany.

Schmidhuber, J. (1991b). A possibility for implementing curiosity and boredom in model-building neural controllers. In Meyer, J.-A. & Wilson, S. W. (Eds.), *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 222–227, Cambridge, MA. Bradford Books/MIT Press.

Singh, S. P. (1991). Transfer of learning across compositions of sequential tasks. In Birnbaum, L. A. & Collins, G. C. (Eds.), *Machine Learning: Proceedings of the Eighth International Workshop (ML91)*, pages 348–352, San Mateo, CA. Morgan Kaufmann.

Smolensky, P. (1988). On the proper treatment of connectionism. *Behavioral and Brain Sciences, 11*(1), 1–23.

Stanfill, C. & Waltz, D. (1986). Toward memory-based reasoning. *Communications of the Association for Computing Machinery, 29*(12), 1213–1228.

Sutton, R. S. (1984). *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, University of Massachusetts, Amherst. Department of Computer and Information Science.

Sutton, R. S. (1986). Two problems with backpropagation and other steepest-descent learning procedures for networks. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 823–831, Hillsdale, NJ. Cognitive Science Society, Lawrence Erlbaum.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*, 9–44.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In Porter, B. W. & Mooney, R. H. (Eds.), *Machine Learning: Proceedings of the Seventh International Conference (ML90)*, pages 216–224, San Mateo, CA. Morgan Kaufmann.

Sutton, R. S. (1991). Reinforcement learning architectures for animats. In Meyer, J.-A. & Wilson, S. W. (Eds.), *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 288–296, Cambridge, MA. Bradford Books/MIT Press.

Sutton, R. S. (Submitted). Adapting bias by gradient descent: An incremental version of Delta-Bar-Delta. GTE Laboratories Inc., 40 Sylvan Rd, Waltham, MA 02254. Submitted to the AAAI-92 conference.

Thrun, S. B. (To appear). The role of exploration in learning control with neural networks. In White, D. A. & Sofge, D. A. (Eds.), *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*. Florence, Kentucky 41022: Van Nostrand Reinhold.

Thrun, S. B. & Möller, K. (To appear). Active exploration in dynamic environments. In *Advances in Neural Information Processing Systems 4*, San Mateo, CA. IEEE, Morgan Kaufmann.

Utgoff, P. E. (1986). *Machine Learning of Inductive Bias*. Boston: Kluwer Academic Publishers.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the Association for Computing Machinery*, *27*(11), 1134–1142. Reprinted in J. W. Shavlik and T. G. Dietterich (Eds.), (1990), *Readings in Machine Learning*, Morgan Kaufmann Publishers:San Mateo, CA.

Vere, S. A. (1980). Multilevel counterfactuals for generalizations of relational concepts and productions. *Artificial Intelligence*, *14*, 139–164.

Waldinger, R. (1976). Achieving several goals simultaneously. In Elcock, E. W. & Michie, D. (Eds.), *Machine Intelligence.* Wiley and Sons.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards.* PhD thesis, King's College, University of Cambridge, Cambridge, England.

Weigend, A. S., Rumelhart, D. E., & Huberman, B. A. (1991). Generalization by weight-elimination with application to forecasting. In Lippmann, R. P., Moody, J. E., & Touretzky, D. S. (Eds.), *Advances in Neural Information Processing Systems 3*, pages 875–882, San Mateo, CA. IEEE, Morgan Kaufmann.

Werbos, P. J. (1990). A menu of designs for reinforcement learning over time. In Miller, W. T., Sutton, R. S., & Werbos, P. J. (Eds.), *Neural Networks for Control*, chapter 3, pages 67–95. Cambridge, MA: Bradford Books/MIT Press.

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

Yee, R. C., Saxena, S., Utgoff, P. E., & Barto, A. G. (1990). Explaining temporal differences to create useful concepts for evaluating states. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 882–888, San Mateo, CA. AAAI, Morgan Kaufmann.