COGNITIVE ECONOMY AND THE ROLE OF REPRESENTATION IN ON-LINE LEARNING

By

David J. Finton

A dissertation submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN – MADISON

2002

Abstract

How can an intelligent agent learn an effective representation of its world? This dissertation applies the psychological principle of cognitive economy to the problem of representation in reinforcement learning. Psychologists have shown that humans cope with difficult tasks by simplifying the task domain, focusing on relevant features and generalizing over states of the world which are "the same" with respect to the task. This dissertation defines a principled set of requirements for representations in reinforcement learning, by applying these principles of cognitive economy to the agent's need to choose the correct actions in its task.

The dissertation formalizes the principle of cognitive economy into algorithmic criteria for feature extraction in reinforcement learning. To do this, it develops mathematical definitions of feature importance, sound decisions, state compatibility, and necessary distinctions, in terms of the rewards expected by the agent in the task. The analysis shows how the representation determines the apparent values of the agent's actions, and proves that the state compatibility criteria presented here result in representations which satisfy a criterion for task learnability.

The dissertation reports on experiments that illustrate one implementation of these ideas in a system which constructs its representation as it goes about learning the task. Results with the puck-on-a-hill task and the pole-balancing task show that the ideas are sound and can be of practical benefit. The principal contributions of this dissertation are a new framework for thinking about feature extraction in terms of cognitive economy, and a demonstration of the effectiveness of an algorithm based on this new framework.

Acknowledgements

Halfway through the Ph.D., I said that when I finished I would know I could do anything. Now I have a humbler attitude. I believe that my completion of this work is the result of God's grace and much answered prayer. I would never have finished this dissertation without the support and encouragement of many people. These deserve my acknowledgement and thanks:

Yu Hen Hu, my long-suffering advisor. I'm grateful for his enthusiasm, intellectual curiosity and continuing interest in new ideas.

My dissertation committee: Peter Andreae, Eric Bach, Rick Jenison and Mark Craven. I'm especially grateful to Peter for his deep insights into feature extraction and his willingness to invest time in me. This dissertation—and this writer—were greatly improved by his guidance.

Robert Shelton, my sponsor at the Johnson Space Center. His advice, encouragement and humor kept the light shining at the end of the tunnel.

NASA-JSC, which awarded me a NASA Graduate Student Researchers Program Fellowship—our tax dollars at work!

Lloyd Smith, who hired me as a programmer in his informatics group, and gave me opportunity to expound on AI in group meetings.

Stephen Dembski, who hired me to write NEXTSTEP code for his music composition research. I'm grateful for his kindness and humor.

Finally, I never would have made it without the support of my family and friends, who laughed with me, commiserated with me, and kept me from taking myself too seriously. Thanks!

David J. Finton, December, 2001

Contents

List of Tables

List of Figures

Chapter 1

Introduction

> Life is like playing a violin solo in public and learning the instrument as one goes on.
>
> —*Samuel Butler*

Intelligence reveals itself in many ways; one of the most impressive is the ability to adapt to unknown or changing environments. Without this capability, one can only respond to situations according to instinct, or by applying a previously-written set of rules. If our machines are to be intelligent servants, they must not require us to give them precisely-defined lists of rules for every possible contingency.

Learning "what to look for" in a new situation is one of the hardest aspects of adaptive behavior. It is also one of the critical differences between experts and novices; the experts are able to ignore irrelevant information and focus on details which have a bearing on the task at hand. For example, a first-year medical student will often reach a wrong diagnosis because he has not learned which symptoms are the important ones, and is misled by irrelevant features—"red herrings." If he looks at a blood smear, he will often miss the pathologic clue — say, a fairly rare myeloblast—because of odd shapes of the much more abundant red cells, which are of no importance (Professor Archie MacKinney, University of Wisconsin School of Medicine, personal communication). An expert chess player can quickly reproduce a chess position taken from a game after a

five-second glance at the board—but only if the pieces are placed according to an actual game. With randomly generated chess boards, the expert does as poorly as a beginner in reconstructing the board (de Groot, 1965). In both examples, the expert knows "what to look for" in order to act intelligently. The important features are those which help him perform the task at hand, even though these features may not be sufficient for some other task (such as reproducing random chess-boards).

Although knowing the important features is critical for intelligent action, it is difficult to learn an effective representation from scratch. Yet this is a critical component of adaptive problem-solving: learning to identify the important features through our interaction with the world. This dissertation attempts to show what is required for such on-line feature extraction to succeed. To answer this question in its most general form would require an analysis of representation and cognition far beyond that possible in a single dissertation. Therefore, this dissertation frames the issues within a particular framework for understanding intelligent action: reinforcement learning.

1.1 The focus of this dissertation

The central question addressed by this dissertation is this: *How can an autonomous agent construct a good representation for its task, as it learns the task?* In other words, how can the agent learn "what to look for?" This question leads to the related questions *What are the characteristics of a good representation?* and *How do these characteristics affect learning?* These are profound, open questions; the aim of this dissertation is simply to indicate new directions for the analysis and to propose some provisional criteria for constructing and evaluating representations.

Because this dissertation is about representation—how representations contribute

to problem-solving, and how we can learn them—it takes some of the other parts of the problem for granted. The analysis will assume that we learn the task by learning a set of action values, which are estimates of the long-term reward which will result from taking various actions. Much previous research has been devoted to showing how an agent can learn these values, and how it can be sure that the values converge. This dissertation builds on previous work by showing the link between representation and action values. It analyzes representations in terms of the "true" action values—the steady-state values that the agent would eventually arrive at after sufficient experience, leaving aside the questions of how these values are learned. The assumption is that a representation has certain fixed properties which will either help or hinder the agent in a particular task. Therefore, the steady-state action values reflect these properties of the representation, independent of the particular learning strategy used to learn them (assuming that it works). The agent may converge upon a set of value estimates for various actions and policies, but whether these final estimates are appropriate to the task depends critically on the representation.

This dissertation attempts to find principled answers to questions of representation. The analysis makes some basic assumptions about the task, but is meant to describe principles which are independent of any single task. Although some of the ideas are illustrated by means of a very simple gridworld task, the reader must bear in mind that the autonomous agent lacks our human understanding of grid problems and treats them the same way it treats any arbitrary task — as a black box providing it with certain sensory inputs, effector outputs, and occasional reward signals.

The remainder of this introduction presents reinforcement learning, and shows how the representation problem arises. At its conclusion, the chapter lists the dissertation's

contributions and outlines the following chapters.

1.2 Reinforcement Learning

1.2.1 A qualitative introduction

In a reinforcement learning task, an agent faces the world without a script; it must explore its environment and experiment in order to learn what to do when it faces different situations. Although it does not have a teacher, the agent does receive rewards or reinforcements from time to time. Unfortunately, there can be long delays between an action and the resulting reward; as a result, the agent faces a difficult *credit-assignment problem* in determining which action was responsible for a negative reward, and what it should have done differently. Learning an effective representation can be a profoundly difficult task in this paradigm.

Reinforcement learning is an important field of study for two reasons. First, it may lead to a better understanding of cognition in general, by providing a quantitative model for the analysis of goal-seeking behavior and intelligent adaptation. This model grounds the values of the agent's actions in the rewards it receives from its environment as a result of the actions it chooses. Second, reinforcement learning studies may show us how to make software more useful by allowing it to adapt to the needs of a task instead of having to be explicitly programmed for each narrowly-defined set of circumstances.

1.2.2 The distinguishing characteristic

The distinguishing characteristic of reinforcement learning is the type of feedback given to the agent—not the type of algorithm or representation employed by the agent. In

reinforcement learning, the feedback to the agent is usually just a number, possibly delayed in time. Thus if we categorize learning problems along a continuum, from rich feedback to sparse feedback, reinforcement learning problems are close to the sparse end of the feedback spectrum. In contrast, most supervised learning problems are at the rich end of the scale: the agent has a teacher which presents it with a series of situations to be classified, along with the correct response for each situation. This means that the agent is shown which situations are important, and it can immediately correct each action by comparing it with the correct response. One way of making the feedback less informative is to merely give the agent a number indicating how useful its response was, instead of telling it what it should have done. Another way of making the feedback more sparse is to allow it to be delayed in time, only given for certain "result" states. In reinforcement learning tasks, the agent typically has to cope with both of these complications. Therefore reinforcement learning is a kind of "trial-and-error" learning with occasional feedback from a critic, rather than from a teacher. Yet there are learning problems with even less feedback, in which the agent associates data points with action outcomes, without any external guidance at all. Although reinforcement learning systems may include mechanisms for assimilating the advice of a teacher (for example, see Maclin and Shavlik, 1996), this dissertation focuses on the representational issues that arise in the basic reinforcement learning problem.

1.2.3 Examples of reinforcement learning tasks

The practical applications of reinforcement learning are tasks where we need a system to adapt to its environment, but find it infeasible to provide the system with a teacher. For some tasks, specifying a complete list of the important situations to be learned may

simply be too tedious or expensive. In this case, it may be more economical to enable the system to detect its current state and its degree of success and failure, and then allow the system to learn to maximize its successes. Tesauro's (1995) TD-Gammon used this approach to learn to play backgammon at the level of world-class human players (but note that TD-Gammon also incorporated some helpful bias in the form of human-supplied feature detectors, which is relevant to the issue of feature extraction, discussed below).

In some cases, it might be impossible to anticipate every possible scenario the system could face. A robot explorer should be robust and flexible in its ability to handle unanticipated situations, especially if it is at a great distance, and there is a significant time lag for commands from a controlling station. This is true of the robotic exploration of our solar system, and a reason why some reinforcement learning capability could be important for such tasks.

Sometimes the optimal strategy for a task is simply unknown; in such cases, we would like the learning system to learn a strategy better than its teachers. For example, Crites and Barto (1996) applied reinforcement learning to a four-elevator, ten-floor system, and produced a control policy which out-performed the elevator motion planning schemes devised by Otis engineers. Another example is the job-shop system of Zhang and Dietterich (1996), which used reinforcement learning to produce better space shuttle pre-launch scheduling than NASA experts.

As computers, networks, and embedded computer control become more complex and more prevalent, reinforcement learning may be a key approach for flexible, robust performance without the expense of explicit programming.

1.2.4 A brief history of reinforcement learning

Modern reinforcement learning descends from two separate research threads: the study of animal learning, and the solution of optimal control problems using value functions and dynamic programming. From the research on animal learning, the field draws its emphasis on learning by trial-and-error. For example, Edward Thorndike's "Law of Effect" described how animals alter their tendency to select particular responses to a situation, according to the strength of the satisfaction or discomfort which results as the animal tries various actions in various situations (Thorndike, 1911).

From optimal control, reinforcement learning gained a mathematical formulation of tasks as Markov decision processes, a characterization of the value of a state in terms of the optimal return function (the expectation of the sum of future rewards if the agent acts optimally from this point onward), and an incremental method of calculating state values. An important distinction is that optimal control methods usually require complete knowledge of the system to be controlled, and this information is typically not available to the reinforcement learner. Instead, in a reinforcement learning task, the agent must learn the task dynamics along with the values of states, as it experiences the results of its actions.

These two research threads came together in the late 1980s, along with a third thread concerning temporal-difference learning. In temporal-difference (TD) methods, we make repeated predictions of the value of some quantity, but we update the system according to error estimates that we get by comparing each estimate with the one which followed it, instead of by comparing each estimate with the final outcome. Temporal-difference learning originated in animal learning psychology with the concept of secondary reinforcers. Examples of work incorporating TD methods include Arthur

Samuel's (1959) famous checkers program, Holland's (1986) Bucket-Brigade algorithm, the Adaptive Heuristic Critic (Sutton, 1984; Barto, Sutton & Anderson, 1983), and Richard Sutton's (1988) presentation of TD(λ) as a general prediction method.

In 1989, Christopher Watkins brought these threads together with his development of Q-learning. In Q-learning, the agent maintains a value for each (state, action) pairing, representing a prediction of the worth of taking that action from the state. (These values were represented by the function Q; hence the name "Q-learning"). The agent updates these action values according to the difference between the values at the current state and the best action value of the resulting state. Thus Q-learning combines a trial-and-error sampling of the problem space with an iterative, temporal-difference method for updating the values. The agent updates these values according to its experience of going from one state to the next, either in the real world, or by hypothetical actions in an internal model of the world.

The temporal-difference method is not the only way of doing reinforcement learning. Other methods, such as Monte Carlo, attempt to correlate actions directly with the eventual rewards without considering the intervening history of other states. What all these methods have in common is that they learn a set of action values, based on the results of the agent's actions in the task. These action values allow the agent to act intelligently by always selecting the action which has the highest potential for future reward (the *greedy policy*). This approach is called *value iteration.* Alternatively, we may learn reinforcement learning tasks by updating policy functions without learning action values; this approach is called *policy iteration.* The analysis of this dissertation is based on the value iteration approach.

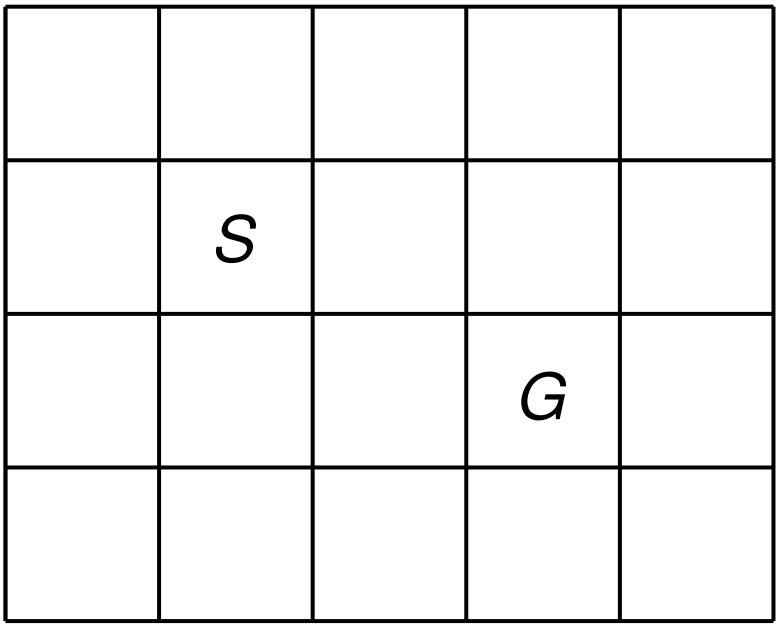


Figure 1: A 5×4 gridworld with start state S (2, 3) and goal state G (4, 2)

1.2.5 An illustration of reinforcement learning

Many of the issues of reinforcement learning, feature extraction, cognitive economy and efficient exploration may be demonstrated by a simple gridworld task. This section shows how an intelligent agent might create a more efficient representation for the task. Although the simplicity of this particular task makes such efforts unnecessary, this exercise brings to light the basic issues, which will be covered in more depth in later chapters. These issues become critical for designing good representations in complex tasks, but the simplicity of the gridworld task makes it a better introduction and demonstration.

The gridworld is shown in Figure 1. The bottom-left cell has coordinates (1, 1) in the grid, and the top-right cell is at (5, 4). In this task, the learning agent starts in state S and its goal is to enter state G as quickly as possible without hitting the walls enclosing the grid. The agent has four possible actions: left, right, up, down. It has no

idea what its goal is, apart from the rewards it receives: when the agent enters state G, it is given a positive reward of $+1$ and the episode ends; when it takes an action which moves it against the enclosing wall it receives a reward of -1, but its position remains the same. Otherwise the agent's reward is 0, and its actions move it one cell across the grid.

Since the agent has no teacher to focus its attention, it learns by stumbling along from its start state until it hits the goal state. If it is extremely lucky, its first attempt may be an optimal path to the goal, say, right, right, down, or down, right, right. More likely, the agent conducts a random walk: it retraces its steps a few times, hits the wall several times, and finally stumbles upon the goal. How can the agent discover a good policy, which produces one of the optimal paths to the goal? This is the problem of reinforcement learning. In general, the agent needs to learn the best action to take from any state it finds itself in. The standard way of doing this is for the agent to learn the action values, as it observes the results of its actions in the grid.

Notice that the goals of the problem depend on the rewards. Changing the reward function changes the nature of the task. For example, we could change the nature of the task from *goal-seeking* to *failure-avoidance* by changing the reward function and list of terminal states: Punish the agent with a reward of -1 whenever it hits one of the outer cells, and otherwise give no reward. (That is, the reward is zero for all actions which leave the agent within the "safe" interior—cells [2, 2], [3, 2], [4, 2], [2, 3], [3, 3], [4, 3]). Now the agent's objective is to avoid the exterior of the grid. If we also specify that the outer cells are terminal but that G is no longer a special terminal cell, the agent's initial episodes will usually result in a failed episode, as it eventually wanders into one of the outer cells. The optimal policy is to wander about in the interior of

the grid. Because most of the agent's initial efforts appear to lead to failure, finding important features can be especially difficult in failure-avoidance tasks. Although the agent never receives positive feedback (it can never "win"), it can eventually learn that some actions postpone failure indefinitely.

In order to know which action to take from a particular state, the agent needs to know which action will lead to the best reward in the long run. The agent can select its actions intelligently by constructing an action value function, $Q(s, a)$, which measures the "goodness" of individual actions. This function should indicate the total reward the agent will receive if it takes action a from state s, acting optimally thereafter. In addition, the function should give higher weight to rewards which come sooner rather than later.

Suppose that as a consequence of taking action a from state s, the agent experiences the reward $r(s, a)$ from the environment and ends up in state s_1. From each successive state s_i, suppose that the agent selects the action with greatest reward, which we will denote a_i^*. The sequence of rewards experienced by the agent is

$$r(s, a), r(s_1, a_1^*), r(s_2, a_2^*), r(s_3, a_3^*), \ldots$$

Let $\gamma < 1$ be a discount factor to be applied to the rewards. We define the action value $Q(s, a)$ as the sum of discounted rewards resulting from taking action a and then choosing the best possible actions for the rest of the episode:

$$Q(s, a) = r(s, a) + \gamma r(s_1, a_1^*) + \gamma^2 r(s_2, a_2^*) + \gamma^3 r(s_3, a_3^*) + \cdots$$

This sum is called a *return.* If action a is optimal, then $Q(s, a)$ is equivalent to the maximum return from s, because the remaining actions, a_i^*, are optimal.

The next chapter develops a more sophisticated definition of action values, in which the rewards and the state transitions may both be stochastic, but this simpler definition

will suffice for a demonstration. We can calculate the action values for our grid task as follows. First, calculate the maximum return possible for each cell. This is easy to calculate because of the following considerations. The only rewards in this task are $+1$ for the actions which lead to G, and -1 for the actions which result in a collision with the wall. From any cell, there is always a path which leads to G without collisions with the wall; therefore, the maximum return will always be positive. Furthermore, on an optimal path the agent will only experience reward on its final move to the goal state. Therefore, the return for an optimal path will simply be the reward $(+1)$, discounted according to its distance into the future. For example, consider the cells which border G on its top, bottom, left and right edges. Since these cells are one step away from G, the agent may move from any of them directly to G, producing a maximum return of $+1$. Next, consider the cells which are one step away from one of the cells bordering G. Since these cells are two actions away from G, their maximum return will be the reward (0) for the action to the cell bordering G, plus γ times the reward $(+1)$ for moving from the border cell to G. This adds up to a return of γ. The cells whose shortest path to G is three steps will have a maximum action value of γ^2, because the only non-zero reward will still be the final one in the path, but this time it is discounted by an extra factor of γ because it is one step further in the future. In the same way, we can calculate the maximum returns for the remaining cells, as shown in Table 1.

γ^4	γ^3	γ^2	γ	γ^2
γ^3	γ^2	γ	1	γ
γ^2	γ	1	G	1
γ^3	γ^2	γ	1	γ

Table 1: Maximum returns

Given the maximum returns, it is easy to calculate the action values for the grid.

According to our definition of $Q(s, a)$, we may rewrite the action value as the sum of the immediate reward for taking action a and γ times the maximum return for the resulting state:

$$Q(s,a) = r(s,a) + \gamma(r(s_1, a_1^*) + \gamma r(s_2, a_2^*) + \gamma^2 r(s_3, a_3^*) + \cdots) = r(s,a) + \gamma R_{s_1}^*$$

where $R_{s_1}^*$ represents the maximum return from s_1. As noted above, if action a takes the agent on an optimal path toward G, then $Q(s, a)$ is the same as the maximum return calculated for s in Table 1. But if a is a bad move, $Q(s, a)$ will be less than the maximum return for s. For example, consider the value of moving `up` from cell (4, 4), which happens to be two cells above G. This action results in a collision with the wall, and a reward of -1. After moving `up`, the agent remains in (4, 4), which has a maximum return of γ. Therefore, $Q((4,4), \texttt{up}) = -1 + \gamma(\gamma) = \gamma^2 - 1$. Table 2 lists the complete set of action values for this task. Although this table does not organize the values according to their placement in the grid, it exemplifies the idea of *discrete representation.* A discrete representation has no high-level features or state-generalization; instead, each state's values are stored separately, with no connection to the values of other states. In the literature, discrete representations are often referred to as tabular (Sutton and Barto, 1998, p. 80) or look-up table representations (Watkins and Dayan, 1992); however, note that we could also have a table look-up representation where the rows of the table refer to groups of states instead of discrete states. For this reason, I will use the term discrete representation.

This example illustrates several important facts about value iteration. First, each action value collapses the entire set of future rewards into the sum of the immediate reward for taking a particular action and the discounted value of the next state. Second, we assume that the action values do not depend on how the agent arrived at its current

s\a	up	right	down	left
(1,1)	γ^3	γ^3	$\gamma^4 - 1$	$\gamma^4 - 1$
(1,2)	γ^4	γ^2	γ^4	$\gamma^3 - 1$
(1,3)	γ^5	γ^3	γ^3	$\gamma^4 - 1$
(1,4)	$\gamma^5 - 1$	γ^4	γ^4	$\gamma^5 - 1$
(2,1)	γ^2	γ^2	$\gamma^3 - 1$	γ^4
(2,2)	γ^3	γ	γ^3	γ^3
(2,3)	γ^4	γ^2	γ^2	γ^4
(2,4)	$\gamma^4 - 1$	γ^3	γ^3	γ^5
(3,1)	γ	γ	$\gamma^2 - 1$	γ^3
(3,2)	γ^2	1	γ^2	γ^2
(3,3)	γ^3	γ	γ	γ^3
(3,4)	$\gamma^3 - 1$	γ^2	γ^2	γ^4
(4,1)	1	γ^2	$\gamma - 1$	γ^2
(4,2)	—	—	—	—
(4,3)	γ^2	γ^2	1	γ^2
(4,4)	$\gamma^2 - 1$	γ^3	γ	γ^3
(5,1)	γ	$\gamma^2 - 1$	$\gamma^2 - 1$	γ
(5,2)	γ^2	$\gamma - 1$	γ^2	1
(5,3)	γ^3	$\gamma^2 - 1$	γ	γ
(5,4)	$\gamma^3 - 1$	$\gamma^3 - 1$	γ^2	γ^2

Table 2: Tabular representation of the action values $Q(s, a)$

state. This assumption—that the identity of the current state alone is sufficient to determine its value, without consideration of the agent's history—is known as the *Markov property.* Third, the agent learns the action values for the task in reverse order. That is, the agent first learns the value of actions which result in immediate rewards, then the values of the actions which led to those states where it receives immediate rewards, then the values of actions which set the stage for those intermediate actions, and so on—from the final states of the task backwards to the starting state. Fourth, the learning is complicated by the need for the agent to explore its environment. The agent lacks our human knowledge of rectangular grids; it does not know which cells result from particular actions without trying them, and it does not even know which direction

to move toward the goal state. The agent only knows its current coordinates on the grid. As far as the agent can tell, there is always a possibility that some un-sampled path may present a "short-cut" to the goal state. For example, moving left from the start state is probably a move in the wrong direction, but could be a smart move if there were some kind of automatic transport from cell $(1, 3)$ to the goal state, $(4, 2)$. Therefore, the agent must explore the grid, even though doing so may appear wasteful in the short-term.

1.3 The representation problem

Action values can be difficult to compute because the values at a particular state depend on all possible paths through future states. As the number of states grows, the number of potential paths through the space explodes.The complexity of computing the action values increases as the square of the number of states (Kaelbling, Littman and Moore, 1996). In addition, the number of distinct values which must be maintained becomes huge; for example, we need a separate row of values for each state in Table 2. In many practical problems the number of states is larger than we would wish for the size of such a table. In particular, any continuous-valued state-space contains an infinite number of states, even if the reachable portion of the space has a small area. As a result, an agent might explore the space indefinitely without twice reaching precisely the same state.

To prevent value iteration from becoming infeasible, the representation must map the actual states of the task onto a smaller number of categories. This results in a manageable number of "generalized states." Whenever the agent needs an action value for a particular state, it references the action value for the corresponding generalized

state or states. The danger is that the values associated with the generalized states are compromises, and if the representation generalizes over states which are too dissimilar, it may lose critical information needed to solve the task. The point is to know what "dissimilar" means in the context of a particular task. By grouping similar states together, the agent no longer has to re-learn the appropriate behavior for each of them separately. Thus the agent requires fewer experiences to learn the task. From the perspective of the agent, the task appears smaller and easier to solve. The trick is to know which states can be grouped together and which distinctions can be safely ignored—and to learn to detect such relationships simply by experiencing the task.

1.3.1 Features, detectors, and generalized states

A *feature* is an attribute which the agent can use to classify its current state. The features used by the agent constitute the set of things that the agent looks for in attempting to describe and understand its situation. For example, in a program which plays chess, the program might characterize the current state in terms of features such as the number of its pieces remaining on the board, the number of the opponent's pieces, whether pieces are threatened or in position to attack, and so on. If the environment simply gives the agent the identity and location of the chess pieces, the agent may need to supplement these low-level features with a set of higher-level features which are functions of the raw inputs.

I use the term *generalized state* to describe a set of states which imply the presence of a particular feature. For example, the top row of our gridworld is a generalized state which covers the individual states (1, 4), (2, 4), (3, 4), (4, 4), and (5, 4). Sometimes it will be convenient to talk about a generalized state in terms of the set of states

which it covers, but sometimes it will be more convenient to give a rule which describes which states are included. I will call such a rule or function a feature, even though it might not be given explicitly in the representation. For example, we can define a function which outputs the value 1 for states in the top row, and otherwise outputs the value 0. This function and the description "top row of the grid" are equivalent rules for the feature "top-row." Although the agent may or may not have a particular *feature detector* which outputs a 1 for states in the top row, we can still analyze the representation in terms of this feature, even though it is not explicitly realized in the agent. Therefore, generalized states and features are both ways of describing the state generalization: generalized states do so in terms of the actual sets of states, and features do so by means of rules describing those sets.

Hence a generalized state is associated with a feature describing its members. Conversely, each feature defines a corresponding generalized state, which is the set of states recognized by that feature. Given a set of binary, non-overlapping features, these generalized states are partition regions. If the features are allowed to have continuous values and overlap, we can still talk about the recognition set for a feature as a generalized state, but with the realization that these sets will then be fuzzy sets which may overlap. The point is that the action values associated with a particular feature are shared by, or generalized over, the states in the corresponding generalized state.

In the case of the discrete representation (for example, Figure 1 and Table 2), we can think of a set of features which each correspond to a particular cell in the grid. Therefore, a discrete representation of the 5×4 gridworld has a set of 20 features. The discrete representation is fine for this task, but the method does not scale to complex tasks and is simply infeasible for continuous state-spaces. State generalization

can result in a smaller representation, in which some of the features correspond to generalized states. The action values are stored on the basis of the features, so that states detected by a particular feature share the same set of action values and tend to be grouped together as "the same kind of thing." If we allow features with continuous membership functions, states may have different degrees of membership in a particular generalized state. (Another interpretation is that the features represent *probabilities* that a particular state is a member of various generalized states). Chapter 3 defines these terms explicitly, and the following chapters use them to analyze the effects of state generalization upon action value. The purpose of state generalization is to transform the agent's problem into a smaller (and we hope, simpler) problem which has the same solution. The principal benefit of such a representation is that the agent may generalize information across similar states, requiring fewer training examples to learn the action values.

Regions of similar states

In order to decide which states should be grouped together, we need to know which states count as "the same thing" in the context of the agent's task. In the goal-seeking formulation of our gridworld task, we can identify the optimal policy for a cell as an action which has the maximum action value for that cell. Since there may be multiple actions which appear optimal, we may also describe a state in terms of its preferred action set, or *preference set.* The preference sets can be useful in categorizing compatible states. For example, in cell (1, 4) the values for actions right and down are both γ^4, but left and up have values of $\gamma^5 - 1$, so the preference set for cell (1, 4) is {right, down}. Now consider cell (2, 4), where the actions right and down have values of γ^3, the

value for left is γ^5, and the value for up is $\gamma^4 - 1$. Thus (2, 4) has the same preference set as (1, 4): {right, down}. We can define regions of cells which share the same preference sets, leading to the representation shown in Figure 2. In this representation, the six

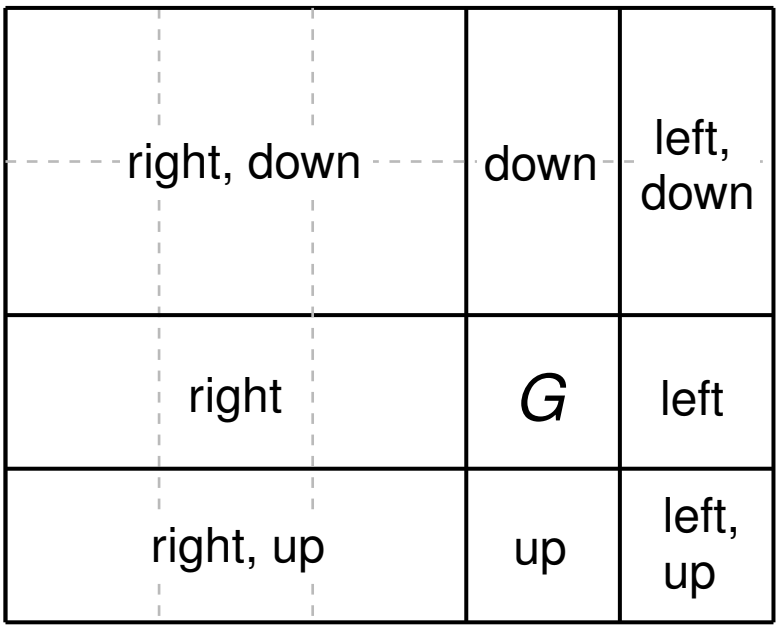


Figure 2: Gridworld partitioned according to preferred actions

cells in the upper left of the grid are grouped together because they share the same preference set: {right, down}. Therefore, we simplify the representation by collapsing them into a single state region. To the right of this group is the group of cells directly above G; the optimal policy for this group is simply to move down. We group the other cells into generalized states in the same way.

Notice that the table of maximum returns (Table 1) does not give us enough information to group states effectively because it does not take the preferred action into account. If we grouped states according to their maximum values, we would put (4, 1) and (4, 3) in the same group, even though one is below G and the other is above G. The agent needs to take different actions from these two cells, moving up from (4, 1),

and down from (4, 3). Therefore, these cells must not be grouped together.

Thus we segregate cells having different preferred actions into different state regions. We would also want to separate cells whose maximum returns differ greatly, even if they have the same preference sets—although this situation does not arise in our simple example. The distinctions between cells within the same region are irrelevant because they tell the agent nothing which will add to its ability to perform the task. For example, knowing that cell (1, 4) is distinct from (2, 4) makes no difference to the agent's policy, which is {right, down} for both these cells. Neither will the slight differences in action values for these cells result in policy differences at earlier states. We learn from this example that some distinctions are not important, but others will prove important if the agent is to perform its task well.

In this way, we may map the complete set of original states onto a smaller set of generalized states. Whether these groupings are appropriate to the task determines whether the agent will meet with success in the task. By choosing good groupings, the agent ignores irrelevant information, while still making whatever distinctions are necessary for solving the task.

How the representation affects learning

By grouping states into regions, the agent is categorizing the world it perceives. In general, the smaller the number of categories, the easier the learning task will be—provided that this categorization preserves all the information needed to perform the task. For any task, we can imagine a continuum of categorization, from very general to very specific. At one extreme, the representation is very simple, but the agent cannot perform the task because it cannot distinguish situations which require different

responses; the agent is ignoring too much information. A simple example would be a gridworld task with a single, large state region for the entire grid. At the opposite end of the continuum, the agent regards every distinct pattern of sensory information as unique; learning is very slow because the agent has to learn the same behavior over and over again for all the different states which might have been grouped together. For example, a discrete representation for a 1000×1000 cell gridworld would result in 1,000,000 cells and very slow learning of the task.

The agent can make its work easier by constructing a representation of the world which is only as fine-grained as it has to be. So the agent should adjust the granularity of the representation in order to make important distinctions where necessary, but otherwise rely on broad general categories. This may mean that the representation makes finer discriminations between states in some areas of the state space than in other areas; therefore, its resolution may vary in granularity. In addition, the representation should be tailored to the task at hand, grounded in the actual rewards that the agent experiences as a result of its actions in that task. The efficiency gained by this kind of *goal-oriented categorization* may be critical for scaling up reinforcement learning to large tasks.

1.3.2 Feature extraction

Feature extraction is the process by which the agent modifies its representation to detect features which are relevant to its task. Since any generalized state corresponds to a feature (which we can think of as a membership function for the states covered by that generalized state), state generalization and feature extraction are merely different ways of looking at the same problem. Therefore, the agent's performance depends critically

on its feature set, since the features are isomorphic to the generalized states. Future chapters will base the analysis of representation and action value in terms of features which describe the generalized states.

If the agent is unable to recognize features that distinguish states requiring different actions, the agent will not be able to perform the task. But if the features separate states which count as "the same kind of thing" in the task, the features represent irrelevant distinctions, and the agent is presented with needless complexity. Ideally, the representation should filter out irrelevant detail, while avoiding over-generalization. Furthermore, when states must be distinguished, some features may do a better job than others at making those distinctions clear; therefore, some features may appear to be more *important* than others.

How can the agent find good features as it learns the task? If we limit ourselves to small tasks which we already understand, we can design a set of feature detectors for the agent. One disadvantage to this approach is that it can involve significant human effort, which may prevent reinforcement learning from scaling up to large tasks or learning tasks which humans have not yet solved. From a research perspective, using hand-designed representations is unsatisfying because it removes the need to understand some of the most important aspects of learning.

The alternative is to automate the process of feature extraction. We want the agent to find important features on its own, grounding its decisions in the actual effects of its actions in the task. The standard approach is to approximate the action values by mathematical functions of the raw state information given by the environment; the agent then tunes these functions by gradient-descent in order to minimize the errors in its prediction of the action values. Gradient-descent techniques tune all the parameters

at once. In effect, this adjusts the value functions for regions of states, as well as the boundaries between those regions.

The gradient-descent approach is often successful, but there are reasons for suspecting that it may be ineffective for some tasks. In order for the value functions to be able to represent the values accurately, they must be tuned so that they do not generalize over states which are too dissimilar. But the error in the current action value does not indicate whether the state generalization is appropriate. For example, suppose that the value functions already generalize correctly over similar states, and put the "boundaries" in the right places. But before learning begins, the predicted values will be inaccurate because the agent has no experience with those states. The errors in the action values will lead the agent to tune the value functions, resulting in changes to the (perhaps implicit) boundaries between generalized states. This is not what was needed.

State generalization usually results in errors in the action values, but those errors are benign when the generalization is done over states which are *compatible.* Unfortunately, gradient-descent cannot distinguish benign errors from harmful ones (caused by generalization over incompatible states, and reflecting an inability to make needed distinctions in the task).

Another current approach is to attempt to ensure that features distinguish states whenever those states have significant differences in any action value. But this approach can make irrelevant distinctions if two states differ on the value of some action which is not a *preferred* action for either state. In that case, the difference is irrelevant, because the agent would never choose that action from either state. For example, in Figure 2 we grouped the cells (1, 4) and (1, 3), even though they have very different values for

the action up: $Q((1,3),\mathsf{up}) = \gamma^5$, while $Q((1,4),\mathsf{up}) = \gamma^5 - 1$. The difference in value for the action up is irrelevant, because up would be a bad action from either state, and would not be chosen.

This dissertation presents a new approach, aimed at finding features which only cover compatible states. It defines state compatibility in terms of the preference sets for the states, so that irrelevant differences are ignored. The goal is to construct a representation according to the same general principles of *cognitive economy* which enable humans and animals to cope with the cognitive demands of difficult tasks.

1.3.3 Cognitive economy and the big picture

Natural intelligences appear to represent the world in ways which filter out irrelevant information and allow them to function in a challenging environment. This seems to hold for both the organization of raw sensory information (perception) as well as higher-level constructs (conception). The resulting categorization of information allows the representation to "provide maximum information with the least cognitive effort," (Rosch, 1978, p. 28).

In the broadest sense, cognitive economy has to do with avoiding irrelevant representational distinctions. The term cognitive economy has also been used in the literature to refer to a particular strategy for information storage in semantic memory nets. To avoid confusion, this section summarizes the earlier work and explains the connection between these two different definitions of cognitive economy. Collins and Quillian (1969) used the term cognitive economy to refer to a principle for eliminating the redundant storage of information. They presented a semantic network model of human memory,

in which facts about different objects are stored at different nodes in a hierarchical network. According to Collins and Quillian, facts which are common to the instances of a category will be stored with the node for the category, rather than being redundantly stored at the nodes for each of the instances. For example, the information that birds can fly would be stored once at the node for "bird," rather than being stored at all the nodes for different kinds of birds. Information about unique traits of a particular object would be stored at the node for that object. Thus accessing category-level facts would require a movement through the hierarchy to the node for the category, resulting in longer reaction times for evaluating the truth of sentences such as "A canary can fly" than for "A canary can sing." Later work by Conrad (1972) supported the claim that words are organized hierarchically in memory, but seemed to show that attributes are stored separately with each instance, rather than once at the level of the superordinate node.

Although Conrad's work seems to be a blow against the idea of cognitive economy, it only applies to the cognitive economy of storage in semantic memory nets. It is based on a model of representation which assumes that instance and category information are stored in separate nodes, and that all possible distinctions are already present in the representation. The question then is whether information is stored in this model in such a way that storage space is conserved at the expense of processing time. Thus Conrad's results do not argue against the more general idea that representing the world efficiently makes us effective in cognitive tasks. In this dissertation, 'cognitive economy' will refer to the more general idea of making relevant distinctions while ignoring irrelevant information.

The phenomenon of *categorical perception* (CP) appears to be an example of this

type of cognitive economy. Stevan Harnad describes categorical perception as "a qualitative difference in how similar things look or sound depending on whether or not they are in the same category" (Harnad, 1987, p. 2). For example, even though color stimuli vary along a smooth continuum of wavelengths, humans break that continuum up into a small set of labelled regions.

> Physically, the wavelength spectrum varies continuously — one wave-length differs from another by simple quantitative change. Psychophysically, human observers can discriminate many wavelengths — our powers to discern are keen. Psychologically, however, hues vary in a categorical fashion—our perceptions cross more or less discretely from one wavelength region to another. Considering hue, brightness, and saturation together, we can tell literally thousands of color nuances apart, but we still partition the color space into relatively few distinct qualitative sensations. (Bornstein, 1987, p. 288)

According to Berlin and Kay (1969), humans break the color spectrum into 11 basic color categories, although languages differ in whether they have basic color terms for all 11. We judge differences between two wavelengths as smaller if they come from the same category, and larger if they come from different categories. For example, we might judge two shades of green as more similar to each other than one of those greens is to a particular shade of yellow, even though the green-yellow comparison may have an equivalent difference in wavelength.

The same effect characterizes perception of speech sounds such as the stop-consonant categories /ba/, /da/, and /ga/. "In other words, in CP there is a quantitative discontinuity in discrimination at the category boundaries of a physical continuum, as

measured by a peak in discriminative acuity at the transition region for the identification of members of adjacent categories" (Harnad, 1987, p. 3). The advantage of this kind of categorization is cognitive economy: "It is more efficient to organize the world into a small number of superordinate units than to deal with each individual exemplar" (Snowdon, 1987, p. 336).

This categorization appears to be the product of both biology and experience. That different human cultures arrive at the same categorical perception of color (Berlin and Kay, 1969) appears to show that some categories are the result of biological constraints. But other evidence indicates that perceptual categories are also the result of experience and learning, resulting in a set of perceptual distinctions which are relevant and diagnostic in the particular cognitive tasks we face. For example, infants that grow up in a particular language environment, say, English, appear to lose the ability to discriminate speech sounds absent in that environment within the first year of life. Werker and Tees (1984) chose two phonetic contrasts which are not present in English, and showed that they could be discriminated by infants from English-speaking homes at six months of age but failed to be discriminated by virtually all of those infants at 12 months. Whether these changes are permanent or not, they are evidence that human perception is organized around distinctions which give us a functional advantage in interacting with our world.

In higher-level tasks, experts appear to learn to represent the relevant details of the environment much more efficiently than novices. In his experiments with chess players, Adriaan de Groot found that the primary difference between master chess players and lesser players was that the masters were able to immediately recognize the important attributes of a chess position. Of the chessmaster, de Groot writes "as a result of his

experience he quite literally 'sees' the position in a totally different (and much more adequate) way than a weaker player [...] His extremely extensive, widely branched and highly organized system of knowledge and experience enables him, first, to recognize immediately a chess position as one belonging to an unwritten category (type) with corresponding board means to be applied, and second, to 'see' immediately and in a highly adequate way its specific, individual features against the background of the type (category)" (de Groot, 1965, p. 306). Thus "[t]he difference in achievement between master and non-master rests primarily on the fact that the master, basing himself on an enormous experience, can start his operational thinking at a much more advanced stage and can consequently function much more specifically and efficiently in his problem solving field" (ibid, p. 307).

To summarize, natural intelligences categorize information in order to reduce the cognitive load of difficult tasks. This categorization is functional, allowing us to ignore irrelevant detail, and to more easily recognize distinctions which are relevant in our world. "It is to the organism's advantage not to differentiate one stimulus from others when that differentiation is irrelevant to the purposes at hand," (Rosch, 1978, p. 29). As a result, the organism is able to perform difficult tasks in spite of its limited cognitive resources.

This dissertation takes the position that artificial intelligences should make use of similar strategies, in order to make the most efficient use of their limited cognitive resources. Specifically, feature extraction should be guided by the principle of cognitive economy. Therefore, a good representation should *not* make all possible distinctions for the tasks we face, but should use broader categories when distinctions are irrelevant to the tasks and rewards we experience. But what does it mean for a feature to be

relevant, and how does that depend on the tasks we want to solve and the nature of our environment? How does our representation change the nature of the apparent task? When should we regard things as "the same"? And how can we learn a good representation of the world through our interactions with it?

1.4 Contributions of This Dissertation

This dissertation translates the idea of cognitive economy into algorithmic criteria for feature extraction in reinforcement learning. To do this, it develops mathematical definitions of feature importance, sound decisions, state compatibility, and necessary distinctions, in terms of the rewards experienced by the agent in its task. It analyzes the connection between the representation and the resulting action values, and offers criteria which ensure that a representation is adequate for the learning of the task. These ideas are explored through theoretical analysis, as well as by implementation and simulation. Contributions include the following:

1. Characterization of learnability in terms of criteria for the allowable loss of reward at each state (*incremental regret*).

2. Definition of feature importance (relevance) in terms of action values, and analysis of the relation between importance and the robustness and efficiency of learning.

3. Characterization of necessary distinctions in terms of learnability.

4. Definition of a set of criteria for state compatibility, and demonstration of the link between these criteria and learnability. (Under certain reasonable conditions, representations which separate incompatible states are proven to meet the

learnability criteria). The compatibility rules specify when states should be considered as "the same kind of thing" in a particular task.

5. Characterization of generalized action values for features and for sets of states, including the role of our prior assumptions in choosing a definition of generalized action values. By defining these values by the steady-state expectations of reward, we may consider the generalized action values to be properties of the representation, apart from any consideration of the convergence of the learning algorithm. Therefore, these definitions of generalized action values show how the representation changes the task perceived by the agent.

6. Successful implementation of an algorithm which generates an effective representation for its task, as it goes about learning the task.

These topics are central to effective learning, because they have to do with the way that state generalization changes the agent's view of the task, whether this makes learning easier or harder, and how the agent may exploit this information to find important features and effective representations, making the most of its limited cognitive resources.

1.5 A Brief Preview of Subsequent Chapters

Chapter 2 formulates the reinforcement learning problem, establishing its essential elements and their mathematical foundation. This provides a basis for considering learning and feature extraction in terms of action values. Along the way, the chapter reviews related work.

Chapter 3 formalizes the principle of cognitive economy into objective criteria expressed in terms of action values. The chapter begins by briefly reviewing cognitive economy from the perspectives of psychologists, economists, and other reinforcement learning researchers. Then it presents the representational model, assumptions, and definitions that form the basis for the rest of the dissertation. Most of the chapter is devoted to the development of criteria for feature importance, sound decisions, necessary distinctions, representational adequacy, and state compatibility.

Chapter 4 extends the definition of action value to generalized states, including those which correspond to continuous-valued, overlapping feature detectors. The chapter indicates how our assumptions about the reinforcement learning problem can lead to different definitions of generalized action values. It applies these definitions to a simple gridworld task, and shows how state generalization changes the agent's perception of the values of its actions.

Chapter 5 analyzes the role that representation plays in allowing the agent to make sound decisions. The chapter builds a link between the previously-derived criteria for compatible states and representational adequacy: partition representations which separate incompatible states are proved to make all necessary distinctions required for the agent to solve the task. This result adds support to the conclusion that the criteria are a well-founded, principled application of cognitive economy to the reinforcement learning problem.

Chapter 6 presents an algorithm which applies the ideas developed in the previous chapters to the solution of two reinforcement learning problems: the puck-on-a-hill, and pole-balancing. The success of the algorithm in constructing an effective representation from scratch demonstrates the potential of a feature extraction strategy which is driven

by considerations of cognitive economy, rather than by the minimization of the top-down errors in action values.

Chapter 7 summarizes the dissertation and gives some thoughts about future work. The Afterword offers a brief consideration of the place of this work within the larger picture of intelligent action.

Chapter 2

Reinforcement Learning and Feature Extraction

> Vast wodges of complex computer codes governing robot behavior in all possible contingencies could be replaced very simply. All that robots needed was the capacity to be either bored or happy, and a few conditions that needed to be satisfied in order to bring those states about. They would then work the rest out for themselves.
>
> —*Douglas Adams, Mostly Harmless*

This chapter formulates the reinforcement learning problem and reviews related work on feature extraction and active learning. The formulation and assumptions presented here are the foundation for the rest of the dissertation. A more comprehensive introduction to reinforcement learning may be found in Kaelbling, Littman and Moore's (1996) survey paper, and in Sutton and Barto's (1998) text.

2.1 The Elements of Reinforcement Learning

2.1.1 The agent and the environment

Reinforcement learning concerns an *agent* which interacts with its *environment* at discrete moments in time; the agent is the learner and decision-maker, while the environment consists of everything outside the agent. (In some literature the agent is referred to as the *controller* and the environment is called the *plant*). The environment may be characterized in terms of the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}_s, \mathcal{P}_r)$, defined as follows:

$\mathcal{S}$ The set of possible states of the environment. $s_t \in \mathcal{S}$ denotes the state at time t.

$\mathcal{A}$ The set of actions available to the agent. $\mathcal{A}(s_t)$ denotes the subset of actions available to the agent in state s_t. We write a_t for the agent's action at time t.

$\mathcal{P}_s(t)$ The state transition probabilities. This function gives the probability of being in state s at time t, based on the past interaction of the agent with the environment. In the most general case, the current state may be a probabilistic function of the entire past history of the agent and the environment. Taking r_t as the reinforcement observed by the agent at time t, we may write

$$\mathcal{P}_s(t) = \Pr\{s_t = s \mid s_{t-1}, a_{t-1}, r_{t-1}, s_{t-2}, a_{t-2}, r_{t-2}, \ldots, r_1, s_0, a_0\}$$

$\mathcal{P}_r(t)$ The reinforcement distribution. The environment gives the agent numerical rewards, $r_t \in \Re$. $\mathcal{P}_r(t)$ is the probability that r is the reward value at time t, based on the previous history of the agent's interaction with the environment. We write

$$\mathcal{P}_r(t) = \Pr\{r_t = r \mid s_{t-1}, a_{t-1}, r_{t-1}, s_{t-2}, a_{t-2}, r_{t-2}, \ldots, r_1, s_0, a_0\}$$

Thus a reinforcement learning task may be specified in terms of its states, actions, and a set of rules which determine how the system dynamics depend on the agent's actions. Any task which can be characterized in this way is a reinforcement learning task. The definition of reinforcement learning has to do with the specification of the task, not with the internal structure or processing of the agent. Usually, the state transition and reinforcement functions are initially unknown to the agent, and must be learned through its interaction with the environment.

The interface between the agent and the environment consists of the inputs to the agent from the environment and the action selection of the agent, which may affect the state of the environment. The inputs to the agent are the current reinforcement, r_t and a signal, $\mathbf{x}_t$, which is a vector of observations (the state-space coordinates) of the current state, s_t.

This dissertation adopts the discrete-time simplification typically assumed in the literature, namely that the agent interacts with the environment at discrete times, $t = 0, 1, 2, 3, \cdots$. At time t the agent is given inputs $\mathbf{x}_t$ and r_t and chooses an action, a_t. The agent observes the results of its action in the environment at time $t + 1$, when its inputs are $\mathbf{x}_{t+1}$ and reinforcement r_{t+1}. Figure 3 shows the relationship between the agent and its environment.

Simply put, the goal of the agent is to maximize its reinforcement over the long run. The challenge of reinforcement learning is for the agent to adjust its behavior as a result of its interaction with the environment, in order to achieve this goal.

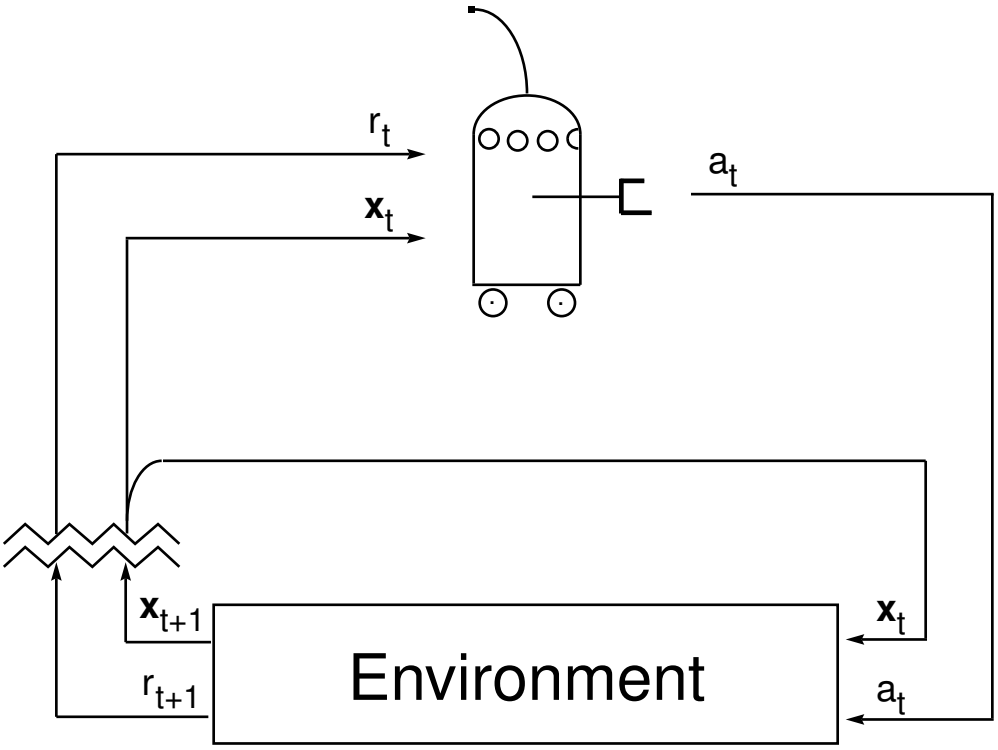


Figure 3: An agent and its environment

2.1.2 Common assumptions

These basic elements define the most general reinforcement learning problem, but most of the research makes some additional assumptions. These include assumptions about the task itself, as well as our criteria for optimal behavior and our standards for judging the success of a reinforcement learning agent. The rest of the dissertation will make the following assumptions.

Number of states and actions In general, $\mathcal{S}$ may contain an infinite number of states, as is the case with tasks having continuous state spaces.

Assume that $\mathcal{A}$ is a small, finite set of actions. Since $\mathcal{A}(s_t)$ is a subset of $\mathcal{A}$, the task may constrain the actions which the agent can take from particular states. For example, a chess task might incorporate a legal-move filter so that the agent only needs to choose among legal moves.

Markov property Assume that the task is Markov: the environment's response to the agent at time $t+1$ depends only on the situation at time t, instead of depending on the entire past history of the agent. Therefore, we may write the transition probability for arriving in state s' as a function of the preceding state and action, ignoring the previous history of the agent:

$$\mathcal{P}^a_{ss'} = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}$$

We could also rewrite the reinforcement probabilities in the same way, but it will be more helpful to deal with the expectation of the reinforcement. Therefore define the expectation of the reinforcement given to the agent after it takes action a from state s and arrives in state s', as follows:

$$\mathcal{R}^a_{ss'} = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\}$$

Assume also that the state observations given to the agent, $\mathbf{x}_t$, are sufficient to uniquely identify the current state s_t without requiring further information about the agent's history; i.e., the task is observable.

Existence of terminal states Assume that the task is *episodic,* having absorbing *terminal states.* This means that the agent's continuing behavior may be broken into a series of episodes, with each episode ending when the agent enters a state which has transitions only to itself (absorbing), and in which the agent only receives reinforcements of zero.

Although some tasks are not episodic, we can sometimes study them by choosing to define certain states as terminal and resetting the system state when the agent enters such a terminal state.

The normal mode of training is to end the episode when the agent arrives at a terminal state, and re-position the agent in a start state for the beginning of a new episode.

Criterion for optimal behavior We will say that the agent acts optimally when it chooses its actions in order to maximize its reinforcement over time—but this could mean different things, depending on how far ahead the agent looks for reinforcement, and how it weighs the value of future reinforcement against current reinforcement.

If the task only lasts for a finite number of steps, we could adopt a *finite horizon* model of optimality, and define the reinforcement over time as the *return*

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T$$

where T is either the index of the last step of the task, or else some fixed number of steps into the future. Then optimal behavior would consist of choosing actions which lead to the highest expected returns.

Instead, this dissertation, assumes the *infinite horizon* model of optimality, in which the return sums up all the reinforcements which follow. We weight the reinforcements by a *discount rate*, $\gamma \in [0, 1]$. Thus

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

The value of γ controls the relative weight of future reinforcement. When $\gamma = 0$, optimal behavior consists of simply selecting the action which is likely to produce the highest immediate reinforcement, without considering the effect that action will have on future reinforcement. For γ between 0 and 1, optimal behavior considers future reinforcements; as γ increases, the optimal agent becomes more farsighted, placing a greater importance upon future reinforcement. Normally, we assume that $\gamma < 1$, unless we know in advance that the task only requires a finite number of steps; this guarantees that the returns are finite, provided that the individual rewards r_t are bounded. Note that stopping the episode at terminal states has no effect on R_t, since all future reinforcements to the agent are zero once it enters a terminal state.

Measuring learning performance It is difficult to assess the worth of a particular learning algorithm without knowing what sort of tasks it is meant to solve, and the reason for learning those tasks. If our goal is to construct a map of the agent's environment and the effects of its actions in all states, we might choose an algorithm which is slow but very accurate. On the other hand, if the agent

needs to quickly adapt to changing conditions, we might care more about the speed of learning than we do about whether the algorithm converges to optimal or near-optimal behavior. It is difficult to characterize convergence speed to near-optimal behavior in a general way.

This dissertation takes the position that the standard for successful reinforcement learning is that the agent learn to act optimally, exerting as little effort as possible. This standard is based on the assumption that not all the information given to the agent is important for enabling it to make the decisions it needs to make in order to perform optimally. By generalizing over similar states, the agent may learn an optimal behavior more efficiently, although this may mean approximating the true values of states and actions at times.

Wisdom is in knowing what details are important, knowing where we must act optimally, and where we can get by with an approximation to the true values of states or actions. This dissertation attempts to demonstrate principled methods of making such judgments, and show that the resulting cognitive economy contributes to faster, more reliable learning.

2.2 Q-Learning

To understand reinforcement learning, we need to see how the agent's representation and its action choices affect the rewards it experiences in the task. To do this, we need to define the values of states and actions in terms of the expectation of future reward. This section develops the basic results for *Q-learning.* In order to develop the theory, it assumes that the state space, $\mathcal{S}$, is finite, since the theoretical results are based on

the assumption that the task is a finite Markov decision process. A later section will discuss some ways in which real tasks often depart from these assumptions, and how these departures change the learning problem. Although this discussion will focus on Q-learning, these ideas also apply, at least in spirit, to the other popular reinforcement learning algorithms, such as SARSA, Adaptive Heuristic Critic, TD(λ) and Q(λ). This material will provide a basis for defining action values for *generalized states* in the next chapter.

2.2.1 Policies and value functions

We may describe the agent's learned behavior as a *policy*, π, which defines the agent's tendency to take various actions from various states. Thus $\pi(s, a)$ is the probability that the agent will choose action a when in state s. At time t

$$\pi(s, a) = \Pr\{a_t = a \mid s_t = s\}$$

According to our assumption of optimal behavior, the agent should learn a policy in which it selects the action with the highest expected return R_t. Most reinforcement learning algorithms do this by learning *value functions*, which predict the return the agent can expect when proceeding from a particular state under a particular policy. Thus we define the value of a state s as the expected return for an agent starting in s at time t, and following policy π:

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\}$$

where $E_{\pi}\{\ldots\}$ denotes the expected value of a quantity, given that the agent follows policy π.

V^π is called the *state-value function for policy* π. We can likewise define the *action-value function for policy* π, written $Q^\pi(s,a)$, which is the expected return for an agent which starts from state s, performs action a, and follows policy π thereafter:

$$Q^\pi(s,a) = E_\pi\{R_t \mid s_t = s, a_t = a\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\}$$

Notice that the only difference between $Q^\pi(s,a)$ and $V^\pi(s)$ is that $Q^\pi(s,a)$ specifies that the agent takes action a as its first step. So we can write $V^\pi(s)$ in terms of $Q^\pi(s,a)$:

$$V^\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(s,a) Q^\pi(s,a) \tag{1}$$

We can also write Q's in terms of V's, since the value of $Q^\pi(s,a)$ is the sum of the immediate reinforcement upon taking action a from state s, and the discounted state-value of the resulting state:

$$Q^\pi(s,a) = E_\pi\left\{r_{t+1}\right\} + \gamma V^\pi(s_{t+1})$$

If $s_{t+1} = s'$, this is $\mathcal{R}^a_{ss'} + \gamma V^\pi(s')$. Taking into account all the possible successor states s' reachable from s, we have

$$Q^\pi(s,a) = \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right] \tag{2}$$

Let π^* denote an optimal policy, under which the agent always selects actions which have the maximum expected return. Therefore, under policy π^*, the state-value (expected return) of any state is at least as great as under any other policy, π. Writing V^* for V^{π^*}, we have

$$V^*(s) \geq V^\pi(s), \text{for all } s \in \mathcal{S}$$

If our task is Markov and has finite state and action spaces $\mathcal{S}$ and $\mathcal{A}$, there will always be at least one such optimal policy; furthermore, if there is more than one optimal

policy, they will share the same optimal state-value function $V^*(s)$ and action-value function $Q^*(s,a)$ (Watkins, 1989).

Under an optimal policy π^*, our previous definition of the state-action values (Equation 2) becomes

$$Q^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^*(s')\right] \tag{3}$$

But notice that we can express $V^*(s')$ in terms of $Q^*(s',a')$. $V^*(s')$ is the expected return from s', assuming that we act optimally. Thus, if a' is an optimal action choice in state s', $Q^*(s',a') = V^*(s')$. Therefore,

$$V^*(s') = \max_{a'} Q^*(s',a'),$$

and, substituting for $V^*(s')$ in Equation 3, we have

$$Q^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s',a')\right]. \tag{4}$$

Equation 4 is known as the Bellman optimality equation for Q^* (Sutton and Barto, 1998, p. 76).

Once the agent has learned the optimal action-value function Q^*, it can easily determine an optimal policy, as follows

$$\pi^*(s,a) = \begin{cases} 1 & \text{if } a = \text{argmax}_{a'}\, Q^*(s,a') \\ 0 & \text{otherwise} \end{cases}$$

(Note: in order to ensure that π is valid as a probability, once we find an optimal action a' for state s, we set $\pi(s,a'') = 0$ if we find any other optimal actions a'').

From Equation 4 we see that $Q^*(s,a)$ is really the expectation of the quantity

$$\hat{r}(s,a) = r + \gamma \max_{a'} Q^*(s',a'),$$

taken over transitions to possible resulting states s' with associated reward r. Therefore, we might think to estimate $Q^*(s,a)$ by a running average of the observations of $\hat{r}(s,a)$. We call the algorithm that does this *one-step Q-learning*. At each step, it updates $Q_t(s,a)$, which is its current estimate of $Q^*(s,a)$, according to the following rule:

$$Q_{t+1}(s_t,a_t) \leftarrow (1-\alpha)Q_t(s_t,a_t) + \alpha\hat{r}(s_t,a_t)$$

where $\alpha \in [0,1]$ is a learning rate parameter. This is usually expressed in the following form:

$$Q_{t+1}(s_t,a_t) \leftarrow Q_t(s_t,a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q_t(s_{t+1},a) - Q_t(s_t,a_t)\right] \tag{5}$$

This algorithm is a form of model-free, off-policy, temporal-difference learning. Model-free refers to the fact that Q-learning does not require the agent to have a complete model of the environment, as dynamic programming methods typically do. Off-policy means that the policy that the agent uses to generate its behavior may be unrelated to the policy which it is estimating. This allows the agent to explore the state-space by selecting actions stochastically, even though it is learning the action values of an optimal policy. Temporal-difference learning methods update an estimate of a value in terms of future estimates of the value; for example, $Q(s,a)$ is an estimate of future reinforcement, and Q-learning updates these estimates by looking at the estimates from the next state it enters, rather than waiting until an episode has finished and then updating all the action values according to the actual returns observed.

2.2.2 An illustration

The following simple task (Figure 4) will serve to illustrate these ideas. In this task, there are three states: $\mathcal{S} = \{x, y, w\}$. State w is an absorbing terminal state. The state

transitions and reinforcement are deterministic, and given in the figure. Thus $\mathcal{A}(x) =$ {right, up}, and $\mathcal{A}(y) =$ {left, up}. Moving right from x always results in state y, and moving left from y always results in state x. These transitions incur a reward of -1. But moving up from either x or y results in state w and the end of the episode. The transition $x \xrightarrow{\text{up}} w$ results in reward of $+10$, and the transition $y \xrightarrow{\text{up}} w$ results in reward of $+100$. We will take the value of the discount rate, γ, to be 0.9.

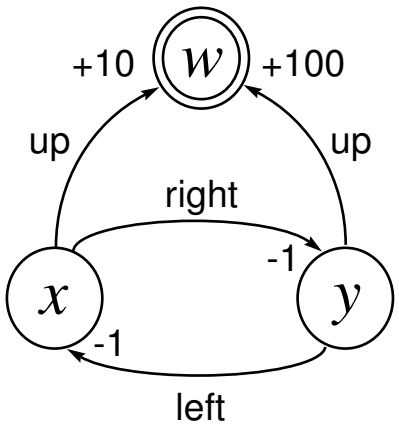


Figure 4: A simple three-node reinforcement learning task

In order to calculate the state values for this problem, we must define a policy, since $V^\pi(s)$ depends on π. Suppose π is defined as follows (with non-existent actions noted by "—"):

π	left	right	up
x	—	0.7	0.3
y	0.8	—	0.2

Working backwards, we may calculate $V^\pi(x)$ and $V^\pi(y)$ in terms of $V^\pi(w)$. To do this, we can combine Equation 1 and Equation 2:

$$V^\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s') \right] \tag{6}$$

This is known as the Bellman equation for V^π (Sutton and Barto, 1998, p. 70).

Ignoring transitions with zero probability ($\pi(s,a) = 0$ or is undefined), we have

$$\begin{aligned} V^\pi(x) &= \pi(x,\mathsf{right}) \left\{ \mathcal{P}_{xx}^{\mathsf{right}} \left(\mathcal{R}_{xx}^{\mathsf{right}} + \gamma V^\pi(x) \right) \right. \\ &\quad + \mathcal{P}_{xy}^{\mathsf{right}} \left(\mathcal{R}_{xy}^{\mathsf{right}} + \gamma V^\pi(y) \right) \\ &\quad \left. + \mathcal{P}_{xw}^{\mathsf{right}} \left(\mathcal{R}_{xw}^{\mathsf{right}} + \gamma V^\pi(w) \right) \right\} \\ &+ \pi(x,\mathsf{up}) \left\{ \mathcal{P}_{xx}^{\mathsf{up}} \left(\mathcal{R}_{xx}^{\mathsf{up}} + \gamma V^\pi(x) \right) \right. \\ &\quad + \mathcal{P}_{xy}^{\mathsf{up}} \left(\mathcal{R}_{xy}^{\mathsf{up}} + \gamma V^\pi(y) \right) \\ &\quad \left. + \mathcal{P}_{xw}^{\mathsf{up}} \left(\mathcal{R}_{xw}^{\mathsf{up}} + \gamma V^\pi(w) \right) \right\} \end{aligned}$$

This is

$$\begin{aligned} V^\pi(x) &= 0.7 \left\{ 0 + 1(-1 + 0.9V^\pi(y)) + 0 \right\} \\ &+ 0.3 \left\{ 0 + 0 + 1(10 + 0.9(0)) \right\} \end{aligned}$$

or

$$V^\pi(x) = 2.3 + 0.63V^\pi(y) \tag{7}$$

Similarly,

$$\begin{aligned} V^\pi(y) &= 0.8 \left\{ 1(-1 + 0.9V^\pi(x)) + 0 + 0 \right\} \\ &+ 0.2 \left\{ 0 + 0 + 1(100 + 0.9(0)) \right\} \end{aligned}$$

hence

$$V^\pi(y) = 19.2 + 0.72V^\pi(x) \tag{8}$$

Solving Equation 7 and Equation 8 for the state values of x and y yields $V^\pi(x) \doteq 26.3$ and $V^\pi(y) \doteq 38.2$.

It is interesting to compare the state values derived under policy π with those resulting from an optimal policy. Consider an optimal policy, π^*:

π^*	left	right	up
x	—	1	0
y	0	—	1

In this case, the Bellman equation for V^π (Equation 6) gives us

$$V^*(x) = -1 + 0.9V^*(y)$$

and

$$V^*(y) = 100 + 0.9(0).$$

Therefore, $V^*(x) = 89$ and $V^*(y) = 100$. As we might have expected, the optimal policy gives the agent a much higher expected return.

But the reinforcement learning agent usually does not know the system dynamics, so it cannot directly calculate the state values as we have just done. Furthermore, even when knowledge of the system dynamics is available, these calculations would quickly become very difficult as the number of states increases, especially when the state transitions and reinforcement are non-deterministic. Therefore, most reinforcement learning agents learn estimates of these values, based on the rewards they receive from the environment. One way to do this is to observe complete episodes and estimate the value of a state by the average of the returns which follow the occurrence of that state. Methods which do this are known as *Monte Carlo* methods because they take averages over random samples of actual returns. In the *first-visit MC* method, the agent considers only returns following the first occurrence of a state. For example, suppose that the agent experiences the following five episodes, with the indicated returns R_0 :

1. $x \overset{\mathsf{right}}{\rightarrow} y \overset{\mathsf{left}}{\rightarrow} x \overset{\mathsf{up}}{\rightarrow} w.$

   $R_0 = 6.2 = -1 + \gamma(-1) + \gamma^2(10)$

2. $y \overset{\mathsf{left}}{\rightarrow} x \overset{\mathsf{right}}{\rightarrow} y \overset{\mathsf{left}}{\rightarrow} x \overset{\mathsf{up}}{\rightarrow} w.$

   $R_0 = 4.58 = -1 + \gamma(-1) + \gamma^2(-1) + \gamma^3(10)$

3. $x \overset{\mathsf{right}}{\rightarrow} y \overset{\mathsf{up}}{\rightarrow} w.$

   $R_0 = 89 = -1 + \gamma(100)$

4. $x \overset{\mathsf{right}}{\rightarrow} y \overset{\mathsf{left}}{\rightarrow} x \overset{\mathsf{right}}{\rightarrow} y \overset{\mathsf{left}}{\rightarrow} x \overset{\mathsf{up}}{\rightarrow} w.$

   $R_0 = 3.122 = -1 + \gamma(-1) + \gamma^2(-1) + \gamma^3(-1) + \gamma^4(10)$

5. $y \overset{\mathsf{left}}{\rightarrow} x \overset{\mathsf{right}}{\rightarrow} y \overset{\mathsf{up}}{\rightarrow} w.$

   $R_0 = 79.1 = -1 + \gamma(-1) + \gamma^2(100)$

In trials 1, 3 and 4 the agent begins in state x, so the return following the first occurrence of state x is R_0. In trials 2 and 5, the first occurrence of x is at time $t = 1$, so the return following the first occurrence of x will be R_1:

$$\begin{aligned} \text{From episode 2} \quad &: \quad R_1 = 6.2 = -1 + \gamma(-1) + \gamma^2(10) \\ \text{From episode 5} \quad &: \quad R_1 = 89 = -1 + \gamma(100) \end{aligned}$$

Thus the agent could average the returns which followed its being in state x, in order to form the estimate $V^\pi(x) = (6.2 + 6.2 + 89 + 3.122 + 89)/5 \doteq 38.7$. The agent could also use this method to determine the action values. In these particular episodes, the first occurrence of (x, right) was always also the first occurrence of x, so that the estimate of $Q^\pi(x, \mathsf{right}) = V^\pi(x) = (6.2 + 6.2 + 89 + 3.122 + 89)/5 \doteq 38.7$. We can expect these estimates to become more accurate as the agent experiences more episodes and thus averages the values over a larger set of returns.

One might expect MC methods to be slow because action values are only updated at the ends of episodes, when complete returns R_t become available; however, we may update the action values for each state visited during the episode. This allows the system to quickly learn action values for "early" states in the task.

In contrast, one-step Q-learning updates a single action value at a time, but it does so immediately; at each step it updates the value for the previous state and action. We can see how this works by considering the same series of episodes for our task. As before, $Q_t(s,a)$ is the agent's estimate at time t of $Q^*(s,a)$, the true action value under an optimal policy. We will take the learning rate $\alpha = 0.1$.

Suppose the initial values $Q_0(s,a)$ are all 0. The first transition of the first episode is $x \overset{\mathsf{right}}{\rightarrow} y$, resulting in a reward of -1; we write this as $(x, \mathsf{right}, y, -1)$. We update $Q_0(x, \mathsf{right})$ by means of Equation 5:

$$\begin{aligned} Q_1(x,\mathsf{right}) &= Q_0(x,\mathsf{right}) + 0.1\left[-1 + 0.9\max_a Q_0(y,a) - Q_0(x,\mathsf{right})\right] \\ &= 0 + 0.1(-1 + 0.9\max\{0,0\} - 0) = -0.1 \end{aligned}$$

All action value estimates other than $Q_0(x,\mathsf{right})$ remain unchanged in Q_1. Next, the agent observes the transition $(y, \mathsf{left}, x, -1)$. So we update

$$\begin{aligned} Q_2(y,\mathsf{left}) &= Q_1(y,\mathsf{left}) + 0.1\left[-1 + 0.9\max_a Q_1(x,a) - Q_1(y,\mathsf{left})\right] \\ &= 0 + 0.1(-1 + 0.9\max\{-0.1,0\} - 0) = -0.1 \end{aligned}$$

Finally, the agent observes the transition $(x, \mathsf{up}, w, 10)$. Hence we update

$$\begin{aligned} Q_3(x,\mathsf{up}) &= Q_2(x,\mathsf{up}) + 0.1\left[10 + 0.9\max_a Q_2(w,a) - Q_2(x,\mathsf{up})\right] \\ &= 0 + 0.1(10 + 0 - 0) = 1 \end{aligned}$$

Therefore, at the end of the first episode, the agent's estimate of the action values is

$Q_3(s,a)$	left	right	up
x	—	-0.1	1.0
y	-0.1	—	0.0

Now consider the second episode. The first transition is $(y, \mathsf{left}, x, -1)$, so we update

$$Q_4(y, \mathsf{left}) = -0.1 + 0.1\,(-1 + 0.9 \max\{-0.1, 1\} - (-0.1)) = -0.1$$

So the estimate of $Q(y, \mathsf{left})$ remains the same. The second transition is $(x, \mathsf{right}, y, -1)$, resulting in the update

$$Q_5(x, \mathsf{right}) = -0.1 + 0.1\,(-1 + 0.9 \max\{-0.1, 0\} - (-0.1)) = -0.19$$

The third transition is $(y, \mathsf{left}, x, -1)$. Hence

$$Q_6(y, \mathsf{left}) = -0.1 + 0.1\,(-1 + 0.9 \max\{-0.19, 1\} - (-0.1)) = -0.1$$

Finally, the fourth transition is $(x, \mathsf{up}, w, 10)$. Thus

$$Q_7(x, \mathsf{up}) = 1 + 0.1\,(10 + 0.9(0) - 1) = 1.9$$

Therefore, after the first two episodes, the agent's estimate of the action values will be

$Q_7(s,a)$	left	right	up
x	—	-0.19	1.9
y	-0.1	—	0.0

Our hope is that under Q-learning, the action values $Q_t(s,a)$ will eventually converge to the true values, $Q^*(s,a)$:

$Q^*(s,a)$	left	right	up
x	—	89	10
y	79.1	—	100

Once these values have been learned, we can easily find the optimal state values by taking $V^*(s) = \max_a Q^*(s,a)$. This yields $V^*(x) = 89$ and $V^*(y) = 100$, as expected.

2.2.3 Theoretical convergence

Watkins and Dayan (1992) proved that Q-learning converges to the optimal action-values with probability 1, under the following conditions.

Given: The task is a finite Markov decision process, meaning that $\mathcal{S}$ and $\mathcal{A}$ are finite sets and the Markov property holds;

... the rewards, r_t, are bounded, meaning that there exists some positive number c for which $|r_t| \leq c$, for all t;

... the action values are represented discretely; that is, the values $Q(s, a)$ are stored separately for each combination of s and a;

... updates are performed according to Equation 5, with discount rate γ such that $0 \leq \gamma < 1$;

... each combination of state and action (s, a) is visited infinitely often, although not necessarily in any particular sequence;

... the learning rate, α_t, is chosen so that $0 \leq \alpha_t < 1$. In addition, α_t decreases in a particular way:

Let $t(i, s, a)$ represent the time of the i^{th} experience of the pair (s, a).

$\sum_i \alpha_{t(i,s,a)} = \infty$ and $\sum_i \alpha^2_{t(i,s,a)} < \infty$, for all s and a.

Then: $Q_t(s, a) \to Q^*(s, a)$, as $t \to \infty$, with probability 1.

2.2.4 Practical considerations

Limited sampling, and the need for exploration

In practice, we usually cannot meet all of the conditions necessary for the convergence proof. Perhaps the most severe restriction is the requirement that we continue to sample each state-action pair forever. According to our definition of optimal behavior, the agent should always choose actions which maximize the cumulative reward it receives, but it cannot do this if it continues to sample all actions, including the bad ones. Thus a purely *exploratory* policy is usually sub-optimal. But a purely greedy policy, in which the agent simply *exploits* its current understanding in order to choose the action with the highest value, is also sub-optimal. For example, in our three-state task, the agent did not experience the transition $y \stackrel{\mathsf{up}}{\rightarrow} w$ in its first two trials. If it continued from this point with a greedy policy, it would always move up from x and would never learn that it can achieve a much higher reward of 100 by means of the transitions $x \stackrel{\mathsf{right}}{\rightarrow} y$ and $y \stackrel{\mathsf{up}}{\rightarrow} w$. Because of the agent's limited experience, the transition $x \stackrel{\mathsf{right}}{\rightarrow} y$ appears sub-optimal. This example shows that in order to learn to optimize its behavior in the long run, the agent may need to choose some actions which seem sub-optimal in the short-term.

The usual way of balancing exploration and exploitation is for the agent to begin with a very exploratory policy and gradually shift to a greedy policy as it acquires more experience. Two common ways of accomplishing this are the *ϵ-greedy* policy and the *soft-max* policy. Under an ϵ-greedy policy, the agent chooses its actions according to a greedy policy most of the time, but now and then it chooses an action at random. The probability that the current action will be chosen randomly is ϵ. In order to make the policy increasingly greedy, we can gradually decrease the value of ϵ with time or

experience (Watkins, 1989, p. 178).

A disadvantage of the ϵ-greedy policy is that its exploratory actions are completely random. But under a soft-max policy, the exploration can be biased toward the actions which appear the most promising, as the agent acquires more experience. One way of implementing the soft-max policy is to choose actions according to a Boltzmann distribution, where the likelihood of any particular action a being chosen in state s is given by the equation

$$\Pr(a|s,\tau) = \frac{e^{Q(s,a)/\tau}}{\sum_{a' \in \mathcal{A}(s)} e^{Q(s,a')/\tau}}$$

By initially setting the *temperature* parameter, τ, to a very large number, the agent begins with an essentially random selection of the available actions. As the temperature is lowered, the agent's choices are weighted in favor of the actions with the highest action values. As $\tau \to 0$, the policy becomes greedy, selecting the action with the maximum action value. This method is related to simulated annealing algorithms (Kirkpatrick, Gelatt, and Vecchi, 1983).

The ϵ-greedy and soft-max policies allow the agent to gradually increase the greediness of its action selection. They do this by decreasing a parameter, ϵ or τ, as time passes, reflecting the agent's increasing confidence in its action values. Instead, we might attempt to calculate an explicit confidence measure. For example, in Kaelbling's (1993a) *interval estimation* algorithm, the agent maintains a confidence interval for each state-action pair. This confidence interval is set according to a statistical measure which takes into account both the action values observed and the amount of experience that the agent has had with that state and action. By selecting the action with the highest upper limit, the agent directs its exploration to the actions which appear the most promising.

Decreasing the learning rate

The assumption that the learning rate, α, decreases with time is necessary if we are to guarantee convergence with a stochastic task. Otherwise, continued noise in the observed reinforcement values might cause the action values to continue to fluctuate. But if the task is deterministic, such a strategy is unnecessary; setting $\alpha \equiv 1$ is sufficient for convergence here (Mitchell, 1997, p. 378). For non-deterministic tasks, the following is an example of a rule for α which satisfies the conditions required by the convergence theorem:

$$\alpha_t = \frac{\alpha_0}{n_t(s,a)}$$

where $n_t(s,a)$ represents the number of times which the agent has tried action a in state s, up to time t.

The need for state generalization

To develop the theory behind Q-learning, we have assumed that the action values may be represented discretely, for example, as a look-up table having separate rows for each state s, with separate column entries $Q(s,a)$ for each action a. Unfortunately, the complexity of Q-learning is on the order of the square of the number of states (Kaelbling, Littman and Moore, 1996, p. 248). For all but the smallest tasks, the discrete representation makes the task infeasible. In particular, if the state-space is continuous, the number of states is infinite, and the agent may never see exactly the same state twice. In order to learn the action values for such tasks, the agent must employ the same strategy which humans use to reduce their information-processing burden: generalize over similar states. But placing the region boundaries correctly is critical if the agent is to solve the task. For example, suppose we have a gridworld of

size 100 by 100, partitioned into state regions as in Figure 5.

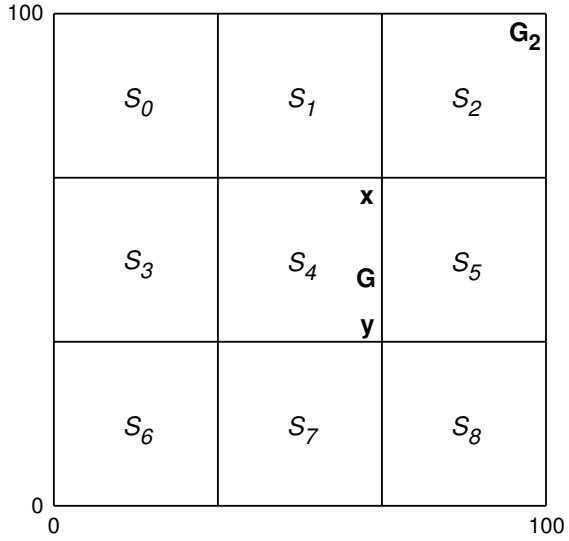


Figure 5: A larger gridworld, with state generalization

States such as x and y would be treated as examples of the same "generalized state," S_4, and would share the same action values, $Q(S_4, \cdot)$, along with any other states which fall in the region S_4.

More generally, we can record the action values as a parameterized function of states and actions, for example: $Q(x, a) = \sum_i w_i f_i(x, a)$, where we call the functions f_i *feature detectors*. To describe the state generalization of Figure 5 in this way, choose the f_i so that each function f_i corresponds to a state region, S_i, where

$$f_i(x, a) = \begin{cases} 1 & \text{if } x \in S_i \\ 0 & \text{otherwise} \end{cases}$$

Since in this example the regions S_i are disjoint (that is, they form a tiling or *partition* of the space), $f_i(x, a) f_j(x, a) = 0$ for $i \neq j$. But in general, the f_i might be any sort of real-valued functions over states and actions.

By grouping certain states together, the agent's representation causes the action values to be generalized over those states. So we adjust the values $Q(S_i, a)$ according to information from any of the separate states covered by the state region S_i. Thus, the apparent value of taking action a in state x will also be affected by the action value updates for the other states in S_4, such as y. All these states share the same action value for a, $Q(S_4, a)$. The success of a particular set of feature detectors will depend on whether the states which are grouped together are actually *similar,* in the sense of having similar action values. Figure 5 is an example of a state generalization scheme which is poorly chosen if our task is to reach the state G. Here the correct policy in state x is $\{\mathsf{down}\}$, while the correct policy in state y is $\{\mathsf{up}\}$. The agent must be able to learn that $Q(x, \mathsf{up}) < Q(x, \mathsf{down})$, but $Q(y, \mathsf{up}) > Q(y, \mathsf{down})$. Therefore, the agent must be able to distinguish between states x and y, and maintain separate action values for these states. Thus x and y should not be grouped together. But notice that if our task is to reach the corner state G_2, the correct policy is $\{\mathsf{up}, \mathsf{right}\}$ for *both* x and y; now it makes sense to group them together. Thus appropriate state generalization depends on the reinforcement expected *under a particular task.*

Watkins (1989) noted that Q-learning may not converge correctly if the action values employ state generalization. Another way of looking at this situation is to realize that, to the agent, a state region S_i may be regarded as a single state whose action values are very noisy, if individual states in S_i have different action values under the discrete state representation. If the learning rate, α, decays appropriately, the action values may, in fact, converge, although not necessarily to values which correspond to an optimal action policy. For example, in the extreme case where we lump all the states into a single state region, the action values will converge—to the expected value of a random

walk in the state-space! This is rarely helpful. Effective learning requires that the agent make appropriate generalizations of states for the task it is learning.

2.3 Survey of Feature Extraction

The lesson of Figure 5 is that representation critically affects the task by the way that it ties together the action values for different states. The central question this dissertation poses is "how can an agent learn which states should be tied together in this way?" This is a question which has been addressed by many other people, but it may help to first describe some of the terminology before seeing how this dissertation fits with related work.

This chapter has introduced the idea of state generalization in Figure 5, by means of a state-space representation which has explicit regions with associated feature detectors, f_i. It seems natural in the context of this system to talk of feature extraction as the process of discovering the f_i. The features and the regions are opposite sides of the same coin, so that state generalization and feature extraction are the same problem.

We have already seen that if we have state-space regions but no features, we can construct features f_i which correspond to the generalized state regions. But what about the reverse situation: the action values are defined in terms of parameterized functions f_i without any explicit specification of state-space regions. These f_i may still tie the action values of certain states together, when those states result in similar patterns of activity in the f_i. For the purpose of analysis, we may call the f_i feature detectors and talk about the corresponding generalized states, even though those generalized states may only be virtual constructions. Thus the generalized state corresponding to the detector f_i would be the set of states for which f_i is active; the common activity of f_i

over these states causes a change in the action values of any one of the states to be reflected in the values of the others.

Therefore, techniques for learning the f_i may be just as relevant to this dissertation as techniques which explicitly change the partitioning of state-space regions. Knowing how to "group states together" is really about knowing when it is appropriate for the action values of different states to be "tied," or mutually dependent.

Most of the related work fits into one of three general categories. The first includes work which attempts to tune feature detectors according to back-propagated error in the action values. The second approach attempts to explicitly change the boundaries of state-space regions, to increase the ability to discriminate states in areas we consider "interesting" or "important." A third approach is to attempt to split regions in order to separate states which show statistical differences in their action values. In contrast, this dissertation introduces a new approach based on the idea of cognitive economy. The essence of this new approach is a decision to ignore differences between states and to group them together, unless distinguishing them can be shown to be important in learning to behave correctly in the task.

2.3.1 Gradient-descent function approximation

Much of the literature uses the term *function approximation* to refer to the process of estimating the action values. Function approximation methods attempt to estimate a function on the basis of its values at sample points. The process of learning the action value function may be considered function approximation because the agent is learning the function $Q(s, a)$, on the basis of the values of Q it observes at individual state transitions.

Gradient-descent—a typical function approximation method—has been a common approach to learning action values in reinforcement learning (Watkins, 1989; Anderson, 1986; Lin,1992; Sutton and Barto, 1998). Sutton and Barto (1998, p. 197) write that "[g]radient-descent methods are among the most widely used of all function approximation methods and are particularly well suited to reinforcement learning." The usual procedure is to first define an error function for the action values, in terms of the difference between the current value and an estimate based on observations from future states. For example, in one-step Q-learning we define the error function

$$\delta_t = \left(r_{t+1} + \gamma \max_b Q_t(s_{t+1}, b)\right) - Q_t(s_t, a_t).$$

We expect that the values obtained at state s_{t+1} will be more accurate than those at state s_t, since they are one step closer to the actual reinforcement from the environment. This is our justification for using the look-ahead values as the training signal for the action value, $Q_t(s_t, a_t)$.

Then we adjust the parameters of the function Q according to a gradient descent in the square of the error. For example, suppose that Q is a linear combination of feature detectors, where each detector f_i has a set of tunable parameters $\mathbf{v}_i = v_{i1}, \ldots, v_{im}$:

$$Q(s, a_j) = \sum_i w_{ij} f_i(s, \mathbf{v}_i)$$

Then we update each of the parameters v_{ik} according to

$$\begin{aligned} v_{ik} &= v_{ik} - \frac{1}{2}\beta \frac{\partial}{\partial v_{ik}}(\delta_t^2) && (9)\\ &= v_{ik} - \beta\delta_t \frac{\partial}{\partial v_{ik}} \left[r_{t+1} + \gamma \max_b Q_t(s_{t+1}, b) \right. && (10)\\ &\quad \left. - \sum_n w_{nj} f_n(s, \mathbf{v}_n) \right] && (11)\\ &= v_{ik} + \beta\delta_t w_{ij} \frac{\partial f_i(s, \mathbf{v}_i)}{\partial v_{ik}} && (12) \end{aligned}$$

where β is a learning rate, $0 < \beta < 1$. The reason that we do not eliminate all the error at once (by setting $\beta = 1$) is that usually there is no single value function which will eliminate all the error for all the states in the region. Therefore, we try to find the best balance by iteratively taking a small step in the direction which will reduce the error most quickly for the current state. As we might expect, the increased cognitive economy of state generalization often comes at the cost of some compromise in the accuracy of the action-value function.

The objective of gradient-descent training is to reduce the mean-squared error (MSE) over some distribution, $\mathcal{P}$, of states and actions:

$$MSE = \sum_t \mathcal{P}(s_t, a_t)\delta_t^2$$

Ideally, $\mathcal{P}$ reflects the actual frequency with which the agent observes states and actions (the *on-policy distribution*). Sutton and Barto (1998) state that it is "not completely clear that we should care about minimizing the MSE" as a performance measure for evaluating function approximation methods, but that "it is not yet clear what a more useful alternative goal for value prediction might be," (p. 196). One advantage of MSE is that it facilitates mathematical analysis, especially for linear gradient-descent methods.

Advantages of gradient-descent methods

Gradient-descent methods capitalize on well-understood techniques for function approximation, such as back-propagation and statistical curve-fitting. By reducing error in the action values, they allow the system to learn $Q(s, a)$ accurately. Gradient-descent techniques learn all components of the action value function at once: the appropriate

"groupings" of states (which may be implicit generalized states), as well as the responsibility of the groups for the overall error.

Disadvantages of gradient-descent methods

There are two principal disadvantages of relying on function approximation to achieve state generalization: the errors driving the process may not be representative of the true error, and errors in the action values are not always relevant to learning the correct behavior for the task.

Gradient-descent techniques are based on the assumptions that the learning system is given a representative sampling of the kinds of states it is to generalize, and that the system is given true error signals. Unfortunately, if the agent explores its environment by selecting actions other than the ones prescribed by an optimal policy, it will be trained on a set of errors for a non-representative sampling of states and actions. Even if the agent proceeds according to a greedy policy, its initial policy is likely to be different than the correct one, again resulting in a non-representative training set for the gradient-descent. A more serious problem may be that the early states of the task see false error signals until the states they lead to are learned. The agent typically learns the final stages of the task first, because they tend to result in direct reinforcement from the environment. After the agent has learned the values of these states, the preceding states will receive accurate updates. In Q-learning, the values get backed up one level for each trial. This means that the values for the first states in the trials are not backed up according to accurate training signals until a significant amount of learning has been done. But the gradient-descent methods use those early, inaccurate updates just the same.

The position of this dissertation is that the accuracy of the values does not always correlate with the accuracy of the policy being learned. If we accept the idea of cognitive economy, then there are some details which are critical to performing the task, and some which are irrelevant. The error-based approach sees errors as errors, without any way of telling which errors are critical and which are benign. To construct a representation which ignores irrelevant detail usually requires that we ignore errors which do not affect the agent's ability to learn the correct policy for the task. Gradient-descent techniques are unable to do this.

2.3.2 Targeting "important" regions in state-space

Another general approach to feature extraction is to focus on regions of state-space which are important in some sense. For example, we may target the regions of state-space which the agent visits most frequently, or areas where the agent's path is most sensitive to its action choices, or areas which are close to simulated trajectories. The agent then adjusts its representation in order to make the finest discriminations among states in these important regions.

Importance is a key concept in this dissertation, but the approach developed here is quite different from the notions of importance proposed by others. This may be seen most clearly by considering a simple task and comparing the requirements of the task with different definitions of importance.

An illustrative example

This section presents a simple gridworld task, displaying the action values and preferred policies throughout its state-space. Applying different feature extraction schemes to a

concrete example will enable us to compare their behavior and judge whether they are able to capture the important distinctions in the task.

Figure 6 shows a 10×10 gridworld in which the agent must choose to move up or right to an adjacent cell at each time-step. The agent's starting state, s_0, is the cell in the lower left corner. The environment returns a reward of $+1$ when the agent enters state G, in the upper right corner of the grid. When the agent moves off the top or right edges of the grid, the environment gives it a reward of -1. Otherwise, the rewards are zero for the agent's moves. The cells which are off the grid are terminal states, as is G; therefore, the agent's current episode ends when it enters one of these cells, and it is placed again at s_0 for the beginning of a new episode.

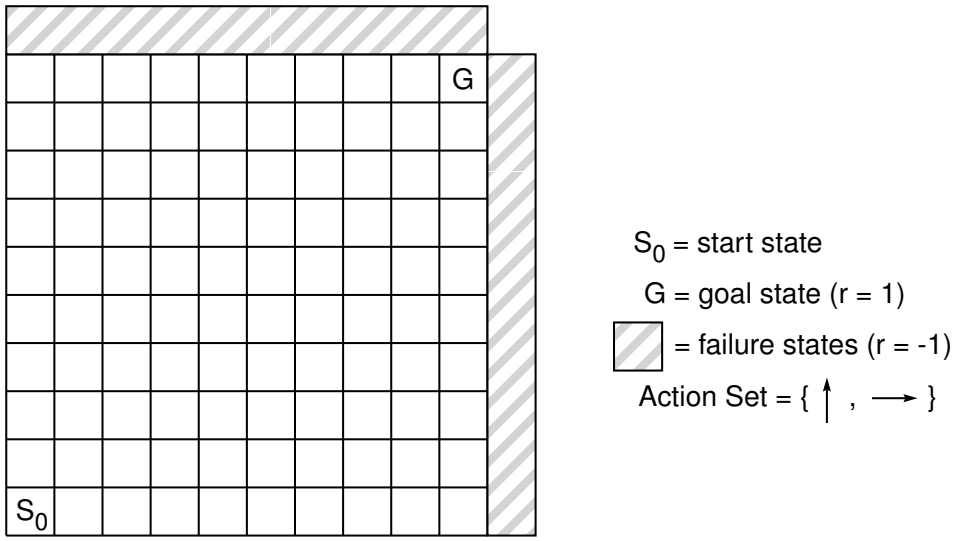


Figure 6: A two-action gridworld task

In this two-action gridworld, there are many paths from the starting state to the goal. In fact, until the agent arrives at either the top row or the right edge of the grid, its choice of action does not matter! The agent needs to move up 10 times and right 10 times in order to reach the goal, but the ordering of these moves does not matter; in

the interior of the grid, either action leads the agent along an optimal path to the goal. Consequently, $Q(s, \mathsf{up}) = Q(s, \mathsf{right})$, unless s is in the top row or the right edge of the grid. For states s in the top row of the grid, moving up is a fatal error but moving right leads toward the goal state, G; thus $Q(s, \mathsf{up}) < Q(s, \mathsf{right})$. The situation is reversed for states on the right edge: here up leads toward success but right leads to a failure state, so $Q(s, \mathsf{up}) > Q(s, \mathsf{right})$. Therefore, the states in the top row require a different policy than the states on the right edge of the grid; these states must not be grouped together.

Representational requirements depend on policy differences

One way to discover the optimal policies for the states is to find the difference of the action values at each state, s: $Q(s, \mathsf{right}) - Q(s, \mathsf{up})$. If this quantity is positive for one state and negative for another, those states have different policies. If this quantity is zero, that means that the agent has the same expectation of success with either action; these "don't care" states may be grouped together with either the up states or the right states.

First, we must compute the action values. The states in the top row and the right edge are the only states which experience direct reinforcement from the environment. They do so by moving to a terminal state, with reinforcement of -1 or 1. Thus for states s in the top row, $Q(s, \mathsf{up}) = -1$, and for s on the right edge, $Q(s, \mathsf{right}) = -1$. For the two states which lead directly to the goal state, G, $Q((9, 10), \mathsf{right}) = 1$, and $Q((10, 9), \mathsf{up}) = 1$. The values of the states $(9, 10)$ and $(10, 9)$ are therefore both 1, since they both allow an action resulting in reward of 1.

We can find the remaining action values by considering the grid as a series of falling

diagonals which each contain the set of states $\{(x, y) : x + y = c\}$ for $c \in [2, 20]$. We observe that all states on the same diagonal have the same value: γ times the value of the states on the next diagonal closer to the goal. For example, the closest diagonal to the goal consists of states (x, y) for which $x + y = 19$. This includes $(9, 10)$ and $(10, 9)$, which both have value 1. Now consider states (x, y) on the diagonal where $x + y = 18$, which is the set $\{(8, 10), (9, 9), (10, 8)\}$. From each of these states, we can reach one of the states in the previous diagonal (where $x + y = 19$) in one move, so these states have a value of $\gamma \times 1 = \gamma$. (We can also move off the grid from $(8, 10)$ and $(10, 8)$, but these fatal moves do not affect the state value because they result in a reward of -1, which is clearly less than the value of the other action). For any state in the next diagonal ($x + y = 17$), each move either results in a state on the preceding diagonal (where $x + y = 18$), giving a value of γ^2, or takes us to a failure state, with value -1. Since we always have an alternative to moving to a failure state, all the states on this diagonal have value γ^2. By continuing in this way, we find that for any state (x, y) on the grid (other than G) and any action a (except for actions leading to failure, which we have already accounted for) $Q((x, y), a) = \gamma^{19-(x+y)}$.

Now that we have calculated the action values, we can plot their differences. Figure 7 shows a plot of $Q(s, \mathsf{right}) - Q(s, \mathsf{up})$ for the two-action gridworld. The values range from -2 (for the right edge) to 2 (on the top row).

Since both actions have the same value for any of the interior states, their value difference is always zero in the interior. We have no need to distinguish these states, since our policy literally does not matter there. The only interesting differences in the action values are at states in the top row, where the differences are all positive, and on the right edge, where they are all negative. Therefore,the agent must distinguish the

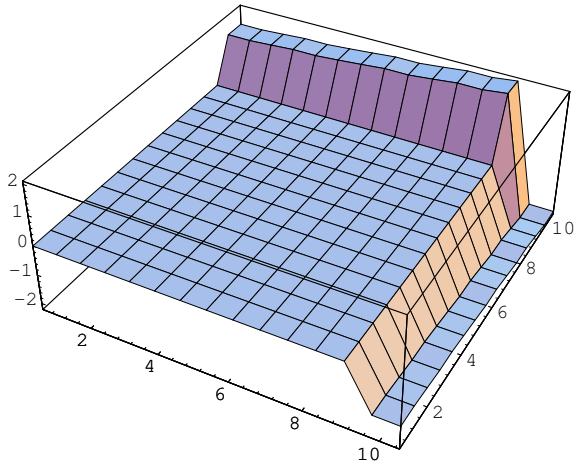


Figure 7: $Q(s, \mathsf{right}) - Q(s, \mathsf{up})$ for the two-action gridworld

states in the top row from the states in the right edge; all other feature information is irrelevant for this task.

Frequency-based feature extraction

In *frequency-based feature extraction,* the agent focuses on features which identify the states it sees most frequently. This is often done by a bottom-up clustering process such as Kohonen's (1990) Self-Organizing Map or Radial-Basis Function tuning (Moody and Darken, 1989; Poggio and Girosi, 1990). The objective is to tune a set of detectors toward the most commonly-visited areas of the state-space, that is, toward peaks in the probability-density function of the state variables. Examples of this approach include work by Holdaway (1989) and Wang and Hanson (1993).

Frequency-based feature extraction is a useful technique for developing a model of the input space, paying special attention to states which are frequently visited. Unfortunately, this strategy is less useful for control tasks, in which the state frequencies

sometimes have no relevance to the decisions the agent must make in the task. Since frequency-based methods ignore the action values, they have no basis for determining when the optimal policy changes between two state regions. Thus they may group together states which have different policies, with the result that the agent will be unable to distinguish those states and will therefore perform incorrectly. In addition, they are sensitive to the agent's exploration policy; if the agent conducts off-policy exploration, the exploratory moves will change the frequencies of the states seen by the agent, and thus change the feature extraction.

The gridworld task shows that the most frequent states are not necessarily the most important in the task. Although frequency may be important in some tasks, the gridworld task is a counter-example to the claim that frequency is an effective measure of feature importance in the general case. Figure 8 plots the probability of entering each state in the course of an episode which begins in s_0, assuming a random action selection by the agent. This plot looks very different from the plot of differences in the action values (Figure 7). The frequency distribution plot emphasizes areas of the grid which are not important in this task: states close to the diagonal and close to the initial state. These are the "don't care" states in our task, but frequency-based feature extraction will attempt to give the agent the highest resolution for these areas of the grid. We could arrive at different frequency distributions if we assume something other than a random policy, but we are unlikely to arrive at a distribution which is relevant to the task without comparing the action values and policies of neighboring states.

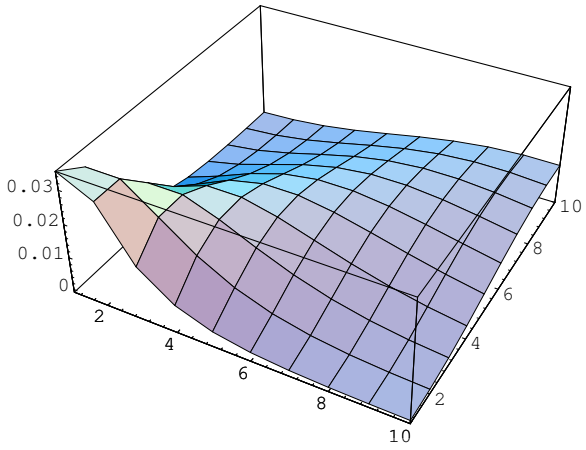


Figure 8: Plot of the state-probability distribution for the two-action gridworld under random exploration

Variable Resolution Dynamic Programming

Andrew Moore's (1991) Variable Resolution Dynamic Programming (VRDP) attempts to "identify areas of state space important to a task," in order "to produce a partitioning with high resolution only in important regions." To do this, VRDP partitions the state space into boxes, indexed by a tree data structure. It learns a model of the environment, which it uses to conduct "mental practice" runs. States visited during mental practice are considered important, and their boxes receive the finest level of partitioning. The result is a representation with a fine resolution along trajectories through state space that correspond to the mental practice. This strategy provides increased resolution for situations the agent must learn to handle, while making broad generalizations over parts of the state-space that the agent never visits.

VRDP makes the assumption that all states along the trajectory are of equal importance, and ought to be represented at the highest resolution. This assumption does

not always hold, and may lead to irrelevant distinctions in control tasks. For example, the last state of the trajectory is the only important one in the gridworld task we have just been considering. VRDP would concentrate on an entire path from s_0 to G, even though the only important areas of the grid are the top and right edges. VRDP also requires the agent to make a good initial guess at a valid trajectory, which prevents the technique from being useful for the general reinforcement learning problem, in which the agent has no way of knowing whether there is a "goal" state, or where it might be. VRDP creates a representation with varying levels of resolution—an important strategy for creating representations with high cognitive economy. For control tasks, the disadvantage of VRDP is that it is blind to the reinforcements, so that it is bound to create many distinctions that are irrelevant to choosing the correct action.

Parti-game

Like VRDP, Moore and Atkeson's (1995) Parti-game algorithm partitions a continuous space, with the greatest resolution about "important" regions; but Parti-game has a more general notion of importance than VRDP. Parti-game assumes that the agent has access to a controller which can guide it in the direction of the goal-state. The state space is partitioned into coarse boxes, and the agent's action at each point is to aim toward a neighboring box. Parti-game assumes that all paths through state space are continuous, so that an action will never take the agent farther than a neighboring box. Therefore, if the agent gets stuck, the agent's clustering must be too coarse to allow the agent to see and navigate around an obstacle at its current position. So it splits the offending box and continues learning. Eventually, the representation has fine enough resolution about the obstacles for the agent to successfully navigate its way to the goal.

Parti-game works well in high-dimensional spaces, if the task can be characterized as a kind of *geometric abstraction task:* the task is deterministic, there is a known region of the space which is the goal, and the actions consist of moving to neighboring regions in the space. But in the general reinforcement learning problem, the agent's actions could lead anywhere, and the states of high reward ("goal states") are only known by the rewards that the agent receives when it lands in those states. Parti-game treats states where the agent's progress is blocked as the important states. This is compatible with a definition of importance in terms of differences of values and policies between states; the agent's failure at these states would typically lead to negative feedback, while some other choice of action would result in success and positive feedback. But the assumptions required for the agent to know where the goal is and when its progress is blocked prevent Parti-game from being a solution to the general problem.

2.3.3 Distinguishing states with different action values

Finally, some work has focused on separating states which differ in the value of one of their actions. Examples include the G-algorithm (Chapman and Kaelbling, 1991) and McCallum's (1995) U-tree and UDM algorithms. Like some of the frequency-based schemes, these methods create a representation of varying resolution, but they do so by attempting to make distinctions precisely where necessary in order to distinguish states which have different action values. Their objective is a representation which allows the agent to accurately predict the expected reward from any state. Therefore, if two states differ on the basis of any of their action values, these methods make a distinction between those states.

The G-algorithm

In the G-algorithm, Chapman and Kaelbling (1991) considered the case where the representation is equivalent to a binary tree, noting that this approach may be extended to the partition case by splitting continuous attribute values at particular thresholds. The G-algorithm splits a region when it determines that an action has a statistically significant difference in value for two states in the region. One limitation of the G-algorithm is that it can only consider a single attribute at a time, and thus cannot separate states on the basis of a complex feature which represents a combination of two or more dimensions of the state-space.

U-Tree

McCallum's U-Tree algorithm (McCallum, 1995) makes *utile distinctions*—distinctions necessary to distinguish states which have different expected rewards. Like the G-algorithm, U-Tree maintains a tree of state-space distinctions, resulting in a set of partitioned regions (the leaves of the tree). Also like the G-algorithm, U-Tree uses a robust statistical test to evaluate whether the values of two adjacent regions are different. U-Tree explores the fringe of the tree, making trial distinctions and then testing whether they help predict future reward. Unlike the G-algorithm, U-Tree is an instance-based algorithm and retains all its observations in memory. It then re-clusters the data for individual states according to its current set of state-space splits, instead of having to start learning the rewards from scratch in a newly-split region. It also is able to evaluate region-splitting criteria based on multiple attributes, unlike the G-algorithm.

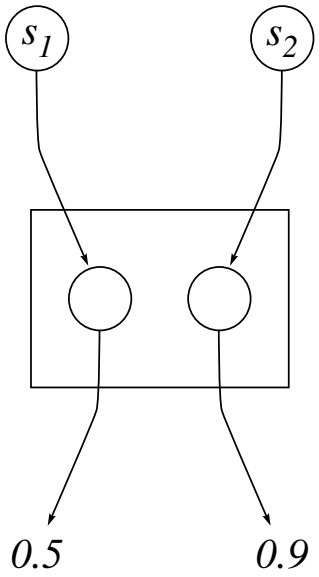


Figure 9: This state region appears to have different values for the action, depending on whether we enter it from s_1 or s_2. Thus the Markov property does not hold for the partition region.

Is the Markov property really the key?

These methods assume that the representation is a partition and that the expected rewards for partition regions should obey the Markov property. Although the task itself is Markov, it might no longer appear Markov to the agent if the representation generalizes over groups of states, as illustrated by Figure 9. Although the Markov property ensures that we may accurately predict future rewards if we know the current state—no matter how we may have arrived in this state—state generalization may prevent the agent from distinguishing individual states in a region. This is the *hidden state* problem, also known as *perceptual aliasing.* One way of detecting the problem is to maintain separate sets of statistics for a region, according to the agent's actions before entering that region. If different paths into the region produce different expectations of reward, it is because the paths lead to aliased states which need to be distinguished.

McCallum's UDM algorithm (McCallum, 1995) uses this strategy to split regions. The other approach, used by the G-algorithm and U-Tree, is to make provisional splits in a region, and make them permanent if there is a statistically-significant difference in expected reward for the sub-regions. In both approaches, we assume that the representation should distinguish states whenever any action has significantly different values for those states.

The approach developed in this dissertation has a different aim: allowing the agent to make *sound decisions* at every state, under the assumption that some differences in action value are not relevant to the agent's choices. The agent does not need the Markov property to hold everywhere in order to learn the best policy. It only needs to be able to predict the values of the states and to choose the correct policy at each state. Some action value differences will have no bearing on either the correct policy or the best expectation of reward from a state. In particular, if two states differ sharply on the value of an action which is not a preferred action for either state, that difference is irrelevant to learning the task. For example, suppose actions a, b, and c have values (1.0, -0.2, 0.3) for state s_1, and have values (0.99, 0.4, 0.3) for s_2. Is this grounds for splitting a region which contains s_1 and s_2? No, because both states have the same preferred action (a), and nearly the same expected reward for taking that action. *The difference in value for action b may be statistically significant, but it has no bearing on the expected return seen by the agent,* because the agent would not choose that action from either state. The Markov criterion requires the representation to distinguish these states so that the agent can accurately predict the values of action b. This example shows that the Markov criterion sometimes leads us to make distinctions which are irrelevant to solving a particular task, because it does not take into consideration whether a

particular distinction is relevant to the decisions the agent must make in the task.

2.3.4 Good representations

What is a good representation? For our gridworld task, a good representation allows the agent to distinguish the top row from the right edge, as simply as possible. Figure 10 shows three successful state partitionings for this task.

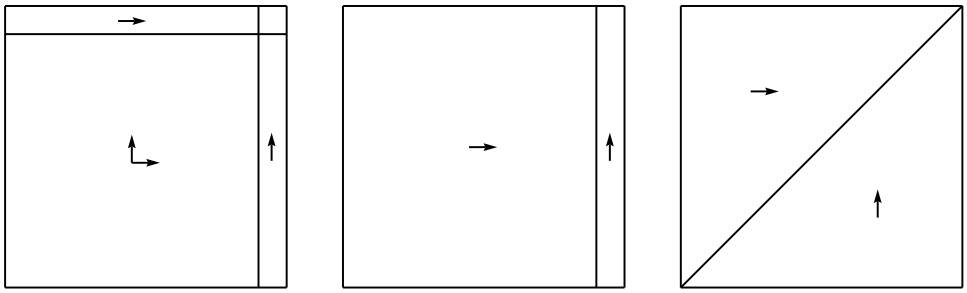


Figure 10: Examples of successful feature extraction for the two-action gridworld task

The first panel shows the grid broken into three regions: the top row, the right column, and the interior of the grid. The arrows show the optimal policy for each of the regions: right for the top row, up for the right column, and either action in the interior. Since either action is optimal in the interior, we could group the interior states with those in the top row, with a policy of right for this region. This arrangement is shown in the middle panel of the figure. (Grouping the "don't care" states with the top row will cause the action-value differences to be less extreme for the top row, but will not have any negative consequences for this task). Even if the action-value differences for a set of states are not zero, if the differences are small enough, it may sometimes still be advantageous to group them together with a more "opinionated" set of states—especially if those differences might be due to noise in the agent's observations. For this reason, we might decide to regard states as "don't care" states whenever the differences

are below some small threshold. Of course, there are many other ways of partitioning the grid. The right panel of the figure shows the grid divided into two regions by the diagonal $y = x$. All three state generalization schemes show great cognitive economy; they reduce the number of states from 100 to two or three. Notice that while there is no one unique optimal representation, the representation must distinguish between states in the top row and states on the right edge. These are the important distinctions for this task, and they arise out of the differences in optimal policy between the states.

How can the agent learn the important distinctions, as it goes about performing the task? We have seen that gradient-descent function approximation can be unrelated to the representational requirements of the task. Frequency-based approaches cannot determine relevant distinctions, because they are blind to differences in the action values. Separating states whenever they disagree on an action value may lead to irrelevant distinctions. What is needed is a way of characterizing action-value differences that affect the agent's ability to make sound decisions in its task. The goal of my dissertation is to meet this need by applying the principle of cognitive economy to the problem of representation. The next chapter begins the analysis with a formalization of cognitive economy for reinforcement learning.

Chapter 3

Formalization of Cognitive Economy for Reinforcement Learning

> At PARC we had a slogan: "Point of view is worth 80 IQ points." It was based on a few things from the past like how smart you had to be in Roman times to multiply two numbers together; only geniuses did it. We haven't gotten any smarter, we've just changed our representation system. We think better generally by inventing better representations; that's something that we as computer scientists recognize as one of the main things that we try to do.
>
> —*Alan Kay*

3.1 Introduction

3.1.1 Overview

Finding a good representation of our world maximizes our ability to function within it. The previous chapter presented the reinforcement learning problem in terms of

the discrete representation, but then showed that most tasks can only be solved if the agent's representation makes use of some form of state generalization. For example, discrete representation of any task having a continuous state-space would result in an infinite number of action values for the agent to learn in order to solve the task.

The most effective representations exhibit high *cognitive economy:* they present the information which is most important for performing the task, and otherwise simplify the task in order to minimize the agent's effort. Current methods of feature extraction tend to either obscure important information or complicate the representation by adding irrelevant information—in both cases, reducing the cognitive economy of the resulting representation. What is needed is a principled way of determining when to distinguish states and when the agent may safely group them together, combining their action values.

The dissertation addresses this need by applying the psychological principle of cognitive economy to the domain of reinforcement learning. Psychologists have shown that humans cope with difficult tasks by simplifying the task domain, focusing on relevant features and generalizing over states of the world which are "the same" with respect to the task. This dissertation defines a principled set of requirements for representations in reinforcement learning, by applying these principles of cognitive economy to the agent's need to choose the correct actions in its task.

3.1.2 State generalization

By allowing the agent to learn action values for entire groups of states, state generalization allows the agent to learn the task with a much smaller number of training

examples—as long as the experiences are generalized over similar states. If the representation groups together states which are "the same kind of thing" in the task, the agent benefits from sharing the experiences of states with other states in the same group. The agent does this simply by storing action values for the *generalized states* which correspond to those groups of states. Knowing how to group the states is critical because bad generalized states will result in attempts to learn policies or values which are incorrect for some of the states. Bad state generalization can make the task harder, or even impossible, to learn.

Figure 11 shows in schematic form how the representation recodes the environment into a form which is more relevant to the agent in its task. To the agent, the representation *is* its environment, since the agent hangs its action values on the framework of generalized states given by the representation. For example, in Figure 11, the representation groups together states s_1 and s_2; therefore, the agent sees them as instances of the same generalized state. This grouping will only be helpful if s_1 and s_2 require the same behavior in the task and are equally desirable states for the agent.

3.1.3 A principled approach

Chapter 2 described previous research concerning representation learning. Most of these efforts appear ad hoc in the context of the reinforcement learning problem because they were not originally developed to meet the needs of a reinforcement learning agent in its task. They attempt to ensure the accuracy or predictability of the action values or they increase the resolution of the state-space about frequently visited states. While these techniques often work, they do not always lead to features which help the agent choose the correct action.

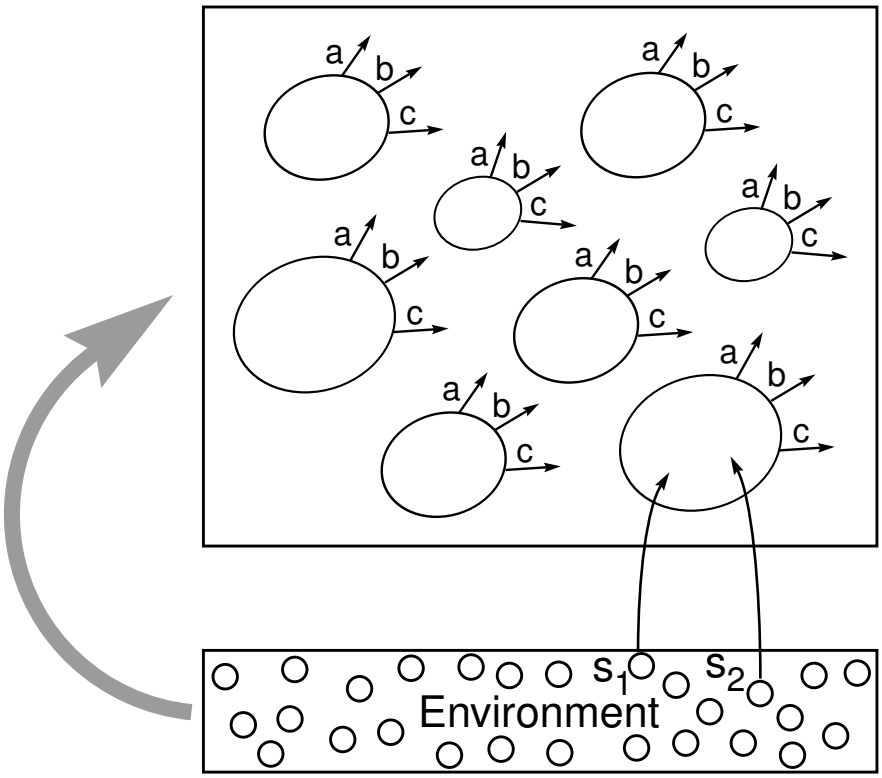


Figure 11: The agent's representation simplifies the environment by grouping states of the world into generalized states that share the same action values

The goal of this dissertation is to develop a more principled approach to representation. The resulting criteria will be principled in the sense that they relate representational distinctions to the ability of an agent to learn to make sound decisions in its task. These criteria build on the principle of cognitive economy, and allow the agent to attend to relevant task details while generalizing over irrelevant information. Since these criteria are tied to the need for the agent to make good decisions in its task, they will focus on important features and necessary distinctions, but may allow other action values to be learned inaccurately. This dissertation takes the point of view that the action values help us learn correct actions, but that accurate action values are not an end in themselves. Learning all the action values to a high level of accuracy is usually a waste of time and effort.

3.2 Cognitive Economy

Cognitive economy generally refers to the combined simplicity and relevance of a categorization scheme or representation. Natural intelligences appear to adopt categorizations with high cognitive economy in order to make sense of the sea of stimuli impinging on their senses without overloading their bounded cognitive resources. Under the heading *Cognitive Economy*, Eleanor Rosch (1978) writes of the "common-sense notion" that the function of categorization is to "provide maximum information with the least cognitive effort," "conserving finite resources as much as possible." (Rosch, 1978, p. 28). She notes that an organism benefits from being able to make as many predictions as possible from the observation of a single property, but that this leads to a huge number of very narrow categories. Then she writes (p. 29):

> On the other hand, one purpose of categorization is to reduce the infinite differences among stimuli to behaviorally and cognitively usable proportions. It is to the organism's advantage not to differentiate one stimulus from others when that differentiation is irrelevant to the purposes at hand.

Cognitive economy results when the representation makes task-relevant distinctions while ignoring irrelevant information. This form of selective generalization presents the agent with a simpler working environment for its task.

In his book Cognitive Economy, Nicholas Rescher expresses this idea in terms of the economic dimension of knowledge—the costs and benefits of acquiring and managing information. He argues that "Cost effectiveness—the proper coordination of costs and benefits in the pursuit of our ends—is an indispensable requisite of rationality," (Rescher, 1989, p. 12). Therefore, a rational agent must assess the cognitive importance of its knowledge: "The assessment of cognitive importance is a key issue for rationality in its economic concern for returns on resource expenditure," (p. 69). Rescher adds (p. 71):

> Importance turns on the extent to which the removal or diminution of a given item would undermine or diminish the prospects of realizing the aims, values, or functions at stake. [...] Accordingly, importance pivots on the idea of making a difference—of casting a large shadow across the particular issues in view.

These common-sense ideas describe mechanisms that allow us to solve difficult tasks in information-dense environments. In order to apply these ideas to the reinforcement learning problem, we will need to explain what it means for a feature to be "relevant to the task," or "make a difference." This chapter defines these ideas in terms of the

agent's action values, which offer an objective measure of its prospects of reaching its goals.

3.2.1 Related ideas in reinforcement learning

Researchers in reinforcement learning recognize the need for cognitive economy, even though they may use different terms to describe it. Regarding the need for generalization, Sutton and Barto (1998, p. 1993) write:

> We have so far assumed that our estimates of value functions are represented as a table with one entry for each state or for each state-action pair. This is a particularly clear and instructive case, but of course it is limited to tasks with small numbers of states and actions. The problem is not just the memory needed for large tables, but the time and data needed to fill them accurately. In other words, the key issue is that of *generalization.* How can experience with a limited subset of the state space be usefully generalized to produce a good approximation over a much larger subset?
>
> This is a severe problem. In many tasks to which we would like to apply reinforcement learning, most states encountered will never have been experienced exactly before. This will almost always be the case when the state or action spaces include continuous variables or complex sensations, such as a visual image. The only way to learn anything at all on these tasks is to generalize from previously experienced states to ones that have never been seen.

Cognitive economy demands that the representation generalize over states, but only when they are "similar," where the meaning of "similar" depends on the agent and its

task. Kaelbling, Littman, and Moore (1996, p. 258) comment:

> In many cases, what we would like to do is partition the environment into regions of states that can be considered the same for the purposes of learning and generating actions.

For most tasks, some areas of the space will require more careful differentiation than others; if the representation allowed for the same level of resolution throughout the state-space, it would be needlessly complex in areas of the space where the task requires less care. Therefore, the smallest representation which makes the distinctions necessary for the task is one which employs a *variable resolution,* focusing its attention and resources on the distinctions which are important to the agent. This is the idea behind Moore's (1991) Variable Resolution Dynamic Programming algorithm, which was designed as a means of increasing the resolution of the representation in "important regions of space." Along the same lines, Hu and Fellman (1996) write: "The algorithm we propose distinguishes important or sensitive sections of a trajectory."

The dissertation builds on these ideas by developing principled criteria for detecting important features and deciding when states are "the same" with respect to the agent's task.

3.2.2 Three aspects of cognitive economy

This dissertation identifies three aspects of cognitive economy that are especially relevant to reinforcement learning: the size of the feature set, the relevance of features to the task, and the preservation of necessary distinctions for success in the task.

If we describe the state of the world in terms of a set of features, we may reduce the size of the feature set by choosing features which generalize over large numbers of states.

The size of the feature set may be easily measured by simply counting the number of features, or by more sophisticated measures such as Minimum Description Length. The need for generalization is a common thread in the literature on representation. The danger of generalization is that inappropriate generalization may prevent relevant features and necessary distinctions from being seen.

Feature relevance and necessary distinctions are the more difficult aspects to define, and are the main focus of this dissertation. Relevant features are properties which make a difference in the agent's task because they identify states where the agent's choice of action is critical to its success in the task. There may be details in the agent's world that are unimportant, because they make little difference in the agent's course of action. A good representation will not only allow the agent to focus on important features, but will capture all the distinctions which are needed to solve the task. This dissertation characterizes important features and necessary distinctions by criteria which are tied to a standard for the learnability of the task.

The next section takes the first step toward formalizing cognitive economy for reinforcement learning by defining a representational model and presenting some assumptions about the agent and its task.

3.3 Preliminaries

For an agent with bounded cognitive resources, state generalization can make the task easier when groups of adjacent states represent "the same kind of thing" in the task. In order to ensure that state generalization is beneficial, we need to make some assumptions about the nature of the agent and its task. The following section makes these assumptions explicit, and defines the basic terminology and system model which will

be common to the rest of the dissertation.

3.3.1 Representational model

In order to analyze the effects of representation upon reinforcement learning we need to know something about the form of the representation. We will assume the following representational model. The action values are represented by a weighted sum of feature detectors, and the weights are updated according to a gradient-descent in the squared error. Thus the action value for the current state and action is given by

$$Q(s, a_k) = \sum_i w_{ik} f_i(s)$$

Each detector, f_i, is a function which maps states to numbers. We associate a weight, $w_{ik} \in \Re$, with the influence of detector f_i upon the value of action a_k. Normally, I will not explicitly indicate the time at which these numbers are taken, but in this section I will index these by a time parameter, in order to consider the updates to the weights. Thus we will refer to the current state as s_t and the current action value as $Q_t(s_t, a_k)$ and the current value of w_{ik} as $w_{ik}(t)$. Then our current action value and weight updates are the following:

$$Q_t(s_t, a_k) = \sum_i w_{ik}(t) f_i(s_t) \tag{13}$$

$$\Delta w_{ik}(t) = \alpha \left(< s_t, a_k > - Q(s_t, a_k) \right) f_i(s_t) \tag{14}$$

Suppose that the agent takes action a_k from state s_t, resulting in state s_{t+1} and reward r_{t+1}. The term $< s_t, a_k >$ represents the reward observed by the agent at time $t + 1$, after executing action a_k in state s_t. The weight update rule comes from the gradient of the squared error of the action value for the current state and action, $Q_t(s_t, a_k)$:

$$\Delta w_{ik}(t) = -\frac{1}{2} \alpha \frac{\partial}{\partial w_{ik}(t)} \left(< s_t, a_k > - Q_t(s_t, a_k) \right)^2 \tag{15}$$

There are several ways of defining the observed value, $< s_t, a_k >$, which is the target for the updates to $Q_t(s_t, a_k)$. One approach is to use the actual return over the whole path, as in Monte Carlo methods:

$$< s_t, a_k >= R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \tag{16}$$

where R_t is only available after a complete episode. Or instead of looking all the way to the end of the episode, we can base the observed value on a one-step look-ahead, as Dynamic Programming methods do:

$$< s_t, a_k >= r_{t+1} + \gamma V^{\pi}(s_{t+1}) \tag{17}$$

where the current estimate of the value of s_{t+1} is

$$V^{\pi}(s_{t+1}) = \sum_a \pi(s_{t+1}, a) Q_t(s_{t+1}, a)$$

and $\pi(s_{t+1}, a)$ represents the probability of taking action a from state s_{t+1} under our current policy, π. The one-step look-ahead rests on the assumption that the information available at time $t+1$ will provide a better estimate than $Q_t(s_t, a_k)$ of the reward for the current episode. Making use of the whole path gives the true value, but requires the agent to wait for the completion of the episode before performing the updates to the function Q. The *temporal difference* methods, TD(λ), provide a bridge between these approaches. In TD learning, the parameter λ is set to a value between 0 and 1; TD(0), also called one-step TD, uses the one-step look-ahead, while TD(1) uses the whole-path return, R_t. The Q-learning algorithm is an off-policy version of one-step TD.

As an aside, note that as long as the f_i are finite, we may assume that their range is $[0, 1]$; that is, $f_i(\mathcal{S}) \subseteq [0, 1]$. If this is not true for a particular feature set, we can always normalize the f_i to be in the range $[0, 1]$, hiding the normalizing factors in the weight

terms. For example, suppose the most general case, in which each detector f_i has range $[a_i, b_i]$. Then we can write the function Q in terms of a normalized set of detectors $\{g_i\}$, where $g_i(s) = \frac{f_i(s)-a_i}{b_i-a_i} \in [0,1]$, and we use appropriate weights $u_{ik} = (b_i - a_i)w_{ik}$. We also introduce a constant-valued detector, $g_c(s) \equiv 1$, with weights $u_{ck} = \sum_i a_i w_{ik}$. In terms of our new, normalized detectors,

$$\begin{aligned} Q(s,a_k) &= \sum_j u_{jk} g_j(s) \\ &= \sum_i u_{ik} g_i(s) + u_{ck} g_c(s) \\ &= \sum_i w_{ik} f_i(s) - \sum_i a_i w_{ik} + \sum_i a_i w_{ik} \\ &= \sum_i w_{ik} f_i(s). \end{aligned}$$

Since the feature detectors have range $[0,1]$, we can think of each detector as an *indicator* of the degree to which its feature describes the current state.

This representational model is specific enough to allow us to examine the effects of state generalization on reinforcement learning; yet it is general enough to cover a wide variety of representations, such as discrete representations, partitions, coarse-coded tilings, and functions of overlapping, continuous-valued detectors. This model is a generalization of Q-learning; we can verify this by showing that its learning rule reduces to that of Q-learning in the case of discrete representations. Suppose we have a dedicated detector for each state:

$$f_i(s_t) = \begin{cases} 1 & \text{if } s_t = s_i \\ 0 & \text{otherwise} \end{cases}$$

Then the value of action a_k is stored separately for each state s_i, since we know by Equation 13 and the definition of f_i that $Q_t(s_i, a_k) = w_{ik}(t) f_i(s_i) = w_{ik}(t)$. Suppose that the agent takes action a_k from state s_i, resulting in state s_j and reward r_{t+1}. Then

Equation 14 takes the form

$$\Delta w_{ik}(t) = \alpha(< s_i, a_k > - w_{ik}(t)).$$

If we take the one-step look-ahead value given by Equation 17, and the agent pursues a greedy policy, this is the familiar weight update for standard Q-learning, but with w's written in place of the Q's:

$$\Delta w_{ik}(t) = \alpha(r_{t+1} + \gamma \max_a w_{ja}(t) - w_{ik}(t)). \tag{18}$$

How does the model generalize to the case where the representation is not discrete? If the f_i are binary detectors which do not overlap, the model takes the form of a partition, in which the state-space is divided into a set of non-overlapping "boxes" (as in the inverted pendulum experiments of Michie and Chambers, 1968, for example). Figure 13a shows a partition representation for the gridworld. Decision trees may also be regarded as partitions, since exactly one leaf or category is active at any given time. As in the case of discrete representations, with a partition there will always be one detector with the value 1 and the rest will all have the value 0. Therefore we again store the value of action a_k for state s_t in a single weight: $Q_t(s_t, a_k) = w_{ik}(t)$, and the update equation is the same as that of the discrete case, given by Equation 18. This is the standard Q-learning update, except that now we are dealing with action values for *regions,* rather than for distinct states. So the weight $w_{ik}(t)$ not only stores the value of action a_k for state s_t, but for other states as well. In effect, the partition regions take the role of *generalized states.*

Finally, we can implement a coarse-coded representation (Hinton, 1984; Albus, 1981) by choosing detectors whose recognition sets overlap; that is, more than one detector will be active for some states. If the detectors are binary, we have an overlapping *tiling* of the space, as illustrated by Figure 13b. In this case the action values are

the sum of the weights $w_{ik}(t)$ for the active detectors. Here our model departs from the update rule for discrete-state Q-learning, since the update for $w_{ik}(t)$ now depends on the contribution of other weights, $w_{jk}(t)$, through the term $Q_t(s_t, a_k)$ (Equation 14). In the most general case, the detectors may have continuous-valued membership functions, as in the example of Figure 13c, and the contribution of $w_{ik}(t)$ to $Q_t(s_t, a_k)$ depends on the strength of detector i's recognition function at the current state, $f_i(s_t)$. Examples of such representations include those which employ perceptron (Rosenblatt, 1962) or radial basis function nodes (Moody and Darken, 1989).

Of course, the action values given by Equation 13 are not necessarily the true action values for state s_t. We must be careful to distinguish between the agent's current estimates, $Q_t(s_t, a_k)$, and the values the agent observes for time t, $< s_t, a_k >$ (whichever definition we use), and the true expectation of reward, which we will denote by $\upsilon^\pi(s_t, a_k)$. The agent observes a sequence of values $< s_t, a_k >$, and learns to estimate these values by its function $Q_t(s, a)$. Whether the function Q actually converges is an important and interesting issue, but one which is beyond the scope of this dissertation. (Please see Watkins and Dayan, 1992, and Singh, Jaakkola and Jordan, 1995, for analyses of the convergence of Q-learning). The true values $\upsilon^\pi(s, a)$ represent the set of targets to which the Q function converges. Thus the function υ does not depend on the training of a particular agent or the convergence of its action values. Instead, it reflects the characteristics of a particular representation, according to our assumptions regarding the observed values and sampling policy. Defining the function υ is non-trivial, and will be left until the following chapter. It will make possible further analysis regarding necessary properties of representations.

This chapter formalizes the principle of cognitive economy into criteria based on

the action values for generalized states—the $w_{ik}(t)$. These *generalized action values* are associated with the reward expected after taking action a_k from a state for which detector f_i is active. The generalized action values apply to features, and will figure prominently in criteria for the importance of features and for necessary distinctions. Since our concern is with the representation, we will assume that these values are the true, steady-state values which will be learned by an algorithm such as those discussed in the previous chapter for learning action values.

3.3.2 Assumptions

Control task

The dissertation assumes that the agent's objective is to learn a specific control task. This assumption has two important implications: that information not relevant to the task may be ignored, and that the agent is forced to choose an action at each step of the problem. Because the agent is facing a specific task, its goal is to learn that task—not to learn a complete map of the action values for the entire state-space. Although a complete mapping of the action values makes optimal performance trivial (since the agent may simply choose the action with the highest value for its current state) learning the best policy does not necessarily require the agent to learn an arbitrarily good mapping of the values. The agent only needs to learn the action values well enough so that it can distinguish the appropriate action at each stage of the problem. Therefore, the agent may be able to make simplifications or generalizations over features and states; in general, this will cause its action values to be less accurate, but only in areas where greater accuracy is not needed for solving the task. A key attribute of control tasks is that the agent is forced to choose an action at each step of the problem.

This *forced-choice property* is the basis for this chapter's definition of the *importance* of a feature in terms of its ability to indicate the advantage of pursuing one action instead of another.

Irrelevant information

Assume that the task permits state generalization and is learnable. Therefore, some of the task information may be ignored by the agent; otherwise, there would be no point in attempting to apply state generalization to the task, since any attempt to generalize over states would lose information vital for performing the task. This dissertation is concerned with tasks in which the important attributes of the state-space may be represented by a simplified model, which will allow the agent to learn the task with less effort—tasks for which the principle of cognitive economy is valid.

Bounded cognitive resources

If the agent had unlimited memory capacity and processing time, representation would be easy; the agent could solve the task by brute force, dividing its state-space into arbitrarily fine divisions. It would regard every configuration of features or sensor data as a unique state. The convergence result for Q-learning with a discrete state space guarantees that the agent will *eventually* find an optimal policy for the task, given sufficient computational effort. Of course, this brute force approach is simply infeasible for most interesting tasks, because real agents have limited memory capacity and processing power.

Human beings face the same obstacles, but cope by ignoring details of the task which are not important to them. Herbert Simon describes human problem-solving as

governed by the *principle of bounded rationality,* which he describes as follows:

> The capacity of the human mind of formulating and solving complex problems is very small compared with the size of the problems whose solution is required for objectively rational behavior in the real world—or even for a reasonable approximation to such objective rationality (Simon, 1957, p. 198).

Therefore, human beings transform the task at hand into a simpler task. Simon writes

> For the first consequence of the principle of bounded rationality is that the intended rationality of an actor requires him to construct a simplified model of the real situation in order to deal with it. He behaves rationally with respect to this model, and such behavior is not even approximately optimal with respect to the real world (ibid, p. 199).

According to Simon, the human response to our bounded rationality is to work with a simplified model of the world in which our solutions will translate into "good enough"—though not optimal—solutions in the real world. Hence we may expect that a reinforcement learning agent will profit from the same strategy of simplifying its representation by ignoring irrelevant detail. This means that the agent generalizes over states which differ only in details which are irrelevant to the agent's task. The trick is in knowing which details are irrelevant and which generalizations would be harmful because they obscure important distinctions between states. Unlike Simon, my hope is that our agent *will* be able to learn to perform its task nearly optimally, with respect to the limited reinforcement defined by the task.

3.3.3 Definitions

Feature A *descriptive property of objects.*

This definition of feature is consistent with Duda and Hart's use of the term to refer to a property which distinguishes objects that belong to different classifications (Duda and Hart, 1973, p. 2). Features may describe a single object, for example, $\mathsf{my\_car}(x)$, ordinary sets of objects, such as $\mathsf{four\_door\_sedan}(x)$ or fuzzy sets, in which the property may be more or less true for different objects, for example, $\mathsf{tall}(x)$.

The objects we will be concerned with are the system states of our agent: $s \in \mathcal{S}$. The classifications we will make have to do with the preferred policies for these states. We are interested in features which allow the agent to learn to classify states according to the actions it must take to maximize reward, and to make these determinations with minimal cognitive effort.

Feature detector *A recognition function of the form* $f : \mathcal{S} \rightarrow [0, 1]$. $f(x)$ *indicates the degree to which the feature is true of the object* x.

If the feature is binary, either $f(x) = 0$ or $f(x) = 1$. For example, the feature $\mathsf{my\_car}(x)$ would be represented by a detector which outputs the value 1 if x is my car, and 0 otherwise. If the feature describes different objects to different degrees, its recognition function will be continuous-valued. For example, we might choose to define a feature detector for the feature $\mathsf{tall}(x)$ by a function like that plotted in Figure 12.

We will use the term *feature detector* to refer to either the *system component* which becomes activated when a particular feature is detected (for example, a hidden

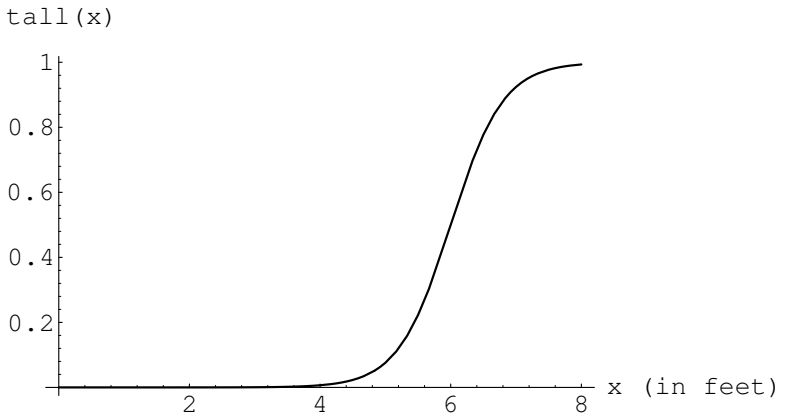


Figure 12: A possible recognition function for the feature tall

node in a neural network) or the *recognition function,* $f(x)$, which describes the behavior of such a component (for example, a sigmoid function which describes the behavior of such a node).

Recognition set *Let* $\mathcal{S}$ *be the set of objects under consideration, and let* $f : \mathcal{S} \rightarrow [0, 1]$ *be a feature detector defined on* $\mathcal{S}$*. Define*

$$\mathcal{S}_f = \{s : s \in \mathcal{S} \text{ and } f(s) > 0\}.$$

We will call $\mathcal{S}_f$ *the recognition set for the feature* f*.*

We will say that the detector *detects* or *recognizes* the objects in its associated recognition set.

Generalized state *The recognition set associated with a feature detector.*

A feature detector f recognizes a set of states (its recognition set), which are described by some feature. The feature is the intension of that set of states, describing their properties. The generalized state is the extension of the feature.

The generalized states, features, and detectors may or may not be defined explicitly by the representation, but they provide the underlying vocabulary for the analysis. At times it will be more convenient to talk about feature detectors and feature extraction; at other times, it will be more convenient to think in terms of the generalized states. The generalized states play a significant role in the analysis to follow, because they describe regions of the state-space which share the same set of action values. By determining the generalized states, the representation determines which action value distinctions will be available to the agent. Calling these regions generalized states highlights their function as entities which

organize the action values, just as ordinary states do in representations that do not generalize.

Preferred action set *Define the preferred action set for a state $s \in \mathcal{S}$ as the set of actions which result in the maximum reward from s:*

$$\text{pref}(s) = \{a_i \in \mathcal{A} : Q^*(s, a_i) = \max_a Q^*(s, a)\}$$

In practice, we may wish to ignore minuscule differences in reward, and include in pref(s) actions which lead to rewards which are "close enough" to the maximum. Given a tolerance, ϵ, we can thus define

$$\text{pref}_\epsilon(s) = \{a_i \in \mathcal{A} : Q^*(s, a_i) \geq \max_a Q^*(s, a) - \epsilon\}$$

Thus $\text{pref}(s) = \text{pref}_0(s)$ and $\text{pref}_0(s) \subseteq \text{pref}_\epsilon(s)$.

Pure and mixed sets of states *Let $S_i \subseteq \mathcal{S}$ be a set of states. We will say that S_i is a* pure *set of states when all the states in S_i share the same preferred action set. That is,*

$$\text{pref}(s_j) = \text{pref}(s_k), \forall s_j, s_k \in S_i.$$

If S_i is not a pure set of states we will call it a mixed *set of states.*

We will be interested in the distinction between pure and mixed sets of states because generalization over mixed sets can obscure important differences in policy. Such generalization may also reduce the apparent relevance, or *importance,* of detectors in a coarse-coded representation. We refer to sets of states with the same ϵ-preference sets ($\text{pref}_\epsilon(s_i)$) as being ϵ-pure.

Some illustrations of features, detectors, pure and mixed sets

Figure 13 shows several different representations for the four-by-four gridworld demonstration presented in Chapter 2. Assume that the start state is the cell $(1, 1)$, the goal state is $(4, 4)$, and let $\mathcal{S}$ represent the set of states, which are the the 16 cells (x, y) in the grid. As in the previous examples, the action set is $\{\mathsf{up}, \mathsf{right}\}$. In part (a) the

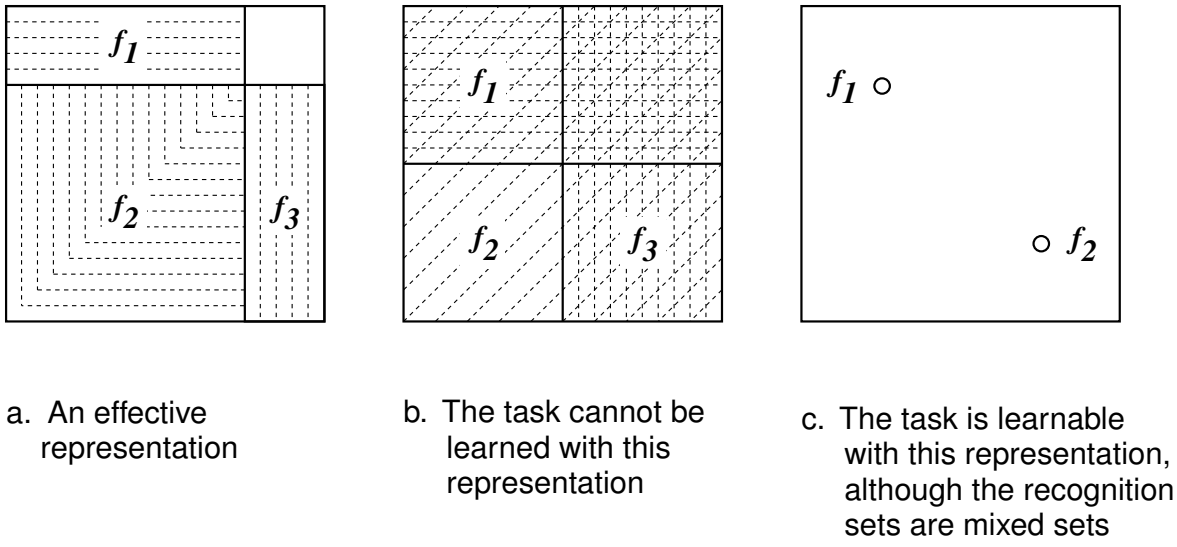


Figure 13: Several different representations of the gridworld

representation is a *partition;* the three detectors are disjoint and collectively cover all of $\mathcal{S}$. (That is, except for the goal state. We may safely ignore the goal state because the episode ends when we enter the goal state; hence, the agent is never called upon to choose an action from this state. Therefore we could extend either f_1 or f_3 to cover the cell $(4, 4)$ without changing any of the action values or policies). The first detector, f_1, covers the top row, except for the goal cell, (4, 4). We can write the recognition function for f_1 as

$$f_1\left((x, y)\right) = \begin{cases} 1 & \text{if } (x, y) \in \{(1, 4), (2, 4), (3, 4)\} \\ 0 & \text{otherwise} \end{cases}$$

The corresponding recognition set is $\mathcal{S}_{f_1} = \{(1,4),(2,4),(3,4)\}$. The preferred action set for all the states in $\mathcal{S}_{f_1}$ is $\{\mathsf{right}\}$; therefore, $\mathcal{S}_{f_1}$ is a pure set. The detector f_3 is similar to f_1, except that its recognition set consists of the cells in the right column of the grid: $\mathcal{S}_{f_3} = \{(4,1),(4,2),(4,3)\}$. Since the preferred action set for all states in f_3 is $\{\mathsf{up}\}$, $\mathcal{S}_{f_3}$ is also a pure set.

The detector f_2 covers the interior cells of the grid; $\mathcal{S}_{f_2} = \{(x,y) : x < 4, y < 4\}$. With a discrete representation, this group of states would appear to be a pure set, since either action is equally good for any of these states. But the groupings of states into $\mathcal{S}_{f_1}$ and $\mathcal{S}_{f_3}$ cause the agent to see a different set of action values in the interior states, and $\mathcal{S}_{f_2}$ will usually be a *mixed set.* How this occurs illustrates some of the effects of state generalization. The key idea is that the agent's action values are shared among all states included in a generalized state. The action values seen by the agent are the ones stored for the generalized states, not the actual values for individual states. Therefore, to the agent, the apparent value of $Q((1,4),\mathsf{right}) = Q((3,4),\mathsf{right}) = Q(\mathcal{S}_{f_1},\mathsf{right})$.

Consider the cell $(1,3)$, which is on the left edge of the grid, and just below the top row. Even though both actions carry the same expectation of reward from $(1,3)$, state generalization makes moving up appear more promising than moving right. Since the agent can only enter the goal state from a state covered by $\mathcal{S}_{f_1}$ or $\mathcal{S}_{f_3}$, the apparent action values at $(1,3)$ depend on how quickly the corresponding actions lead the agent to either of these regions. In cell $(1,3)$, the difference between moving up and moving right is that moving up immediately takes the agent into the generalized state $\mathcal{S}_{f_1}$, which appears to set the stage for success. Moving right simply delays the agent's entry into either $\mathcal{S}_{f_1}$ or $\mathcal{S}_{f_3}$. This delay appears to be a wasted move; the value of entering $\mathcal{S}_{f_1}$ appears to be the same, whether the agent enters the region from $(1,3)$ or from $(2,3)$

or $(3, 3)$. And since the payoff for entering $\mathcal{S}_{f_3}$ is the same as that for $\mathcal{S}_{f_1}$, moving right, right, right to $\mathcal{S}_{f_3}$ appears worse than the prospect of immediately entering $\mathcal{S}_{f_1}$. By this reasoning, we see that the action up looks most promising in all the states to the left of the rising diagonal; the action right appears to have higher value for that part of $\mathcal{S}_{f_2}$ to the right of the diagonal. Thus the way in which the representation groups future states distorts the agent's perception of its current options, and the interior states now differ in their preferred action. Even so, this representation is preferable to the discrete representation, because the reduced representation will usually allow the agent to learn the task in a smaller number of training episodes.

In part (b) of the figure, the detectors are again binary (meaning that their value is either 0 or 1), but the representation is not a partition, because the detectors overlap. Here the detector f_1 recognizes all cells in the top two rows, so the recognition set $\mathcal{S}_{f_1}$ is $\{(x, y) : y > 2\}$. The detector f_3 detects all cells in the right-most two columns; hence $\mathcal{S}_{f_3} = \{(x, y) : x > 2\}$. Detector f_2 recognizes every cell in the grid, so $\mathcal{S}_{f_2} = \mathcal{S}$. Although cells in the lower left quadrant of the grid will only be detected by f_2, the cells in the top right quadrant activate all three detectors. Unfortunately, none of the detectors can distinguish (3, 4), in which the agent's preferred action should be right, from (4, 3), in which the preferred action should be up. Therefore, all three detectors have mixed recognition sets. Since this representation can only remember a single policy for the right quadrant, it will always make mistakes on either (3, 4) or (4, 3), which have no overlap in their preference sets. As a result, the agent will be unable to completely learn the task, no matter how much training it receives.

The representation in part (c) is made up of two Gaussian-kernel detectors:

$$f_i(s) = \frac{e^{-(s-c_i)^2/2\sigma^2}}{e^{-(s-c_1)^2/2\sigma^2} + e^{-(s-c_2)^2/2\sigma^2}}$$

where $(s - c_i)^2$ gives the square of the distance between the current state, s, and the center of detector i. Here $\mathcal{S}_{f_1} = \mathcal{S}_{f_2} = \mathcal{S}$. Even though both detectors have mixed recognition sets, f_1 responds more strongly to cells in the top row, while f_2 responds more strongly to cells in the right column. Hence this representation allows the agent to distinguish cells in which it should move up from cells in which it must move right. This allows the agent to learn the task.

3.4 Relevant Features

If the agent is to construct a representation which focuses on relevant features, it must have some criterion for assessing their importance. Moore (1991) considered the small section of state-space experienced by the agent in practice runs as important, with the unvisited areas of the space unimportant. Hu and Fellman (1996) considered states close to the initial state to be most important, because those states were the most critical for controlling their task (the inverted pendulum). In both cases, the definition of importance had something to do with states which "made a difference" for completing the task. This idea is the key for generalizing the notion of importance for other tasks.

3.4.1 Importance

If a feature makes a difference in the task, it will be relevant to the decisions the agent must make. Therefore, when an important feature is active, the values of different actions will tend to be widely-separated, as illustrated in Figure 14. The generalized state on the left is the recognition set for an unimportant feature, while the generalized state on the right is important. Important features are informative, showing clearly

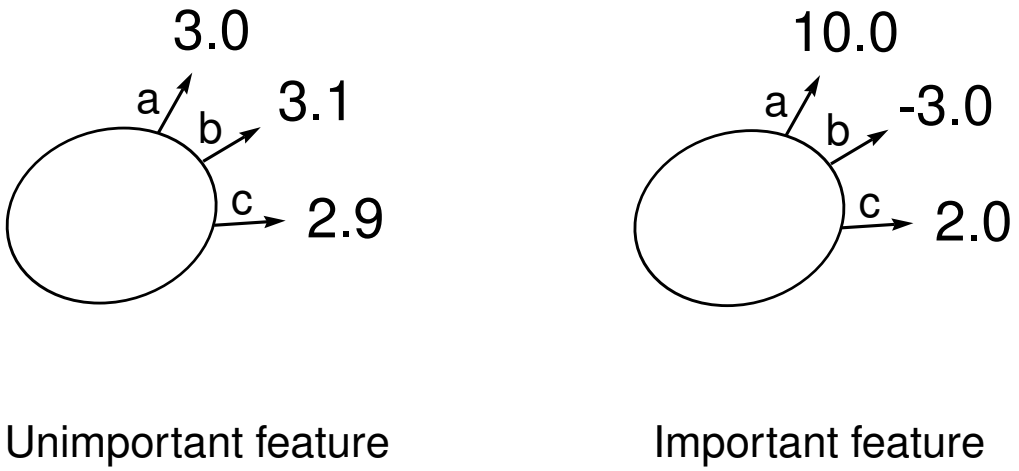


Figure 14: Widely-separated action values are characteristic of important features.

which action the agent should take. They tend to make learning more robust, because their widely-separated values need not be learned as accurately for the agent to discern the preferred action. The generalized state on the right makes a relevant distinction in the task. The generalized state on the left provides little help to the agent in deciding which action to take. It might be the result of bad state generalization; generalizing over a mixed set of states can cause the action values to be averaged over states which have opposite preferences, resulting in "mushed," uninformative values. Or the agent's policy for those states might simply have very little consequence to its overall expectation of success. In either case, the feature (or equivalently, the state-space grouping) is not informative; it is irrelevant to the agent's job of selecting the action which has the best expected reward.

To extract maximum information with the least effort, the agent would like its representation to focus on important features. Widely-spread action values indicate features that reliably indicate significant consequences for the agent's choice of action. My earlier work on *importance-based feature extraction* (Finton and Hu, 1994 and 1995),

explored algorithms which tune feature detectors in order to increase this measure of importance in the detectors. For example, in a neural network, an unimportant hidden node can be deleted without affecting the network's outputs. Such nodes may be reused by moving them to other parts of the state-space, where they may show themselves to be important.

3.4.2 Definitions of importance

One of the previous studies applied importance-based feature extraction to the pole-balancing problem (Finton and Hu, 1994). Since the task had only two actions (push left or push right), that study simply defined the importance of feature detector f_j by

$$I(f_j) = \frac{|w_{j1} - w_{j2}|}{2}$$

To generalize the definition of importance to tasks with larger action spaces, importance should have the following basic properties:

1. Importance is a function of the differences in action values.

2. Importance is zero when all the action values are the same, and nearly zero for don't-care states or features. We will define *don't care* to mean that all actions are in the preferred action set for that state or feature. Therefore, there may be some differences in action values for a don't-care state, although these differences will be negligible, below some error tolerance, ϵ.

3. Importance is high if one action has a significantly higher expectation of reward than the others. Such features are highly informative, and indicate critical decision points in the task.

One candidate for a generalized definition of importance is the difference in value between the best action and the closest non-preferred action. This definition captures the ability of the feature to distinguish the best action from the other actions, but has the disadvantage that it may be sensitive to the parameter ϵ, which describes the membership of the preference set.

Another candidate definition is the standard deviation of the action values for the feature. This definition has the disadvantage of being influenced by very low values which correspond to actions that are nowhere near the preference set.

These definitions are useful places to begin; arriving at a generalized definition of importance is left for future work.

3.5 Necessary Distinctions

The representation should not only focus on relevant features, but it must capture all the *necessary distinctions* needed for the agent to solve its task. By generalizing over states, the representation filters out information; the necessary distinctions are the information which must be preserved in the representation if the agent is to be able to learn to make sound decisions in the task. An *adequate representation* places states in separate categories if they require different actions or if their values must be distinguished for the agent to learn the task. Ideally, the representation should provide these necessary distinctions while focusing on important features and generalizing over irrelevant information. Such representations have a high degree of cognitive economy, and provide the agent with a goal-oriented categorization of its world.

This section presents two types of necessary distinctions: *policy distinctions* and *value distinctions.* Then the section explains what it means for the agent to learn to

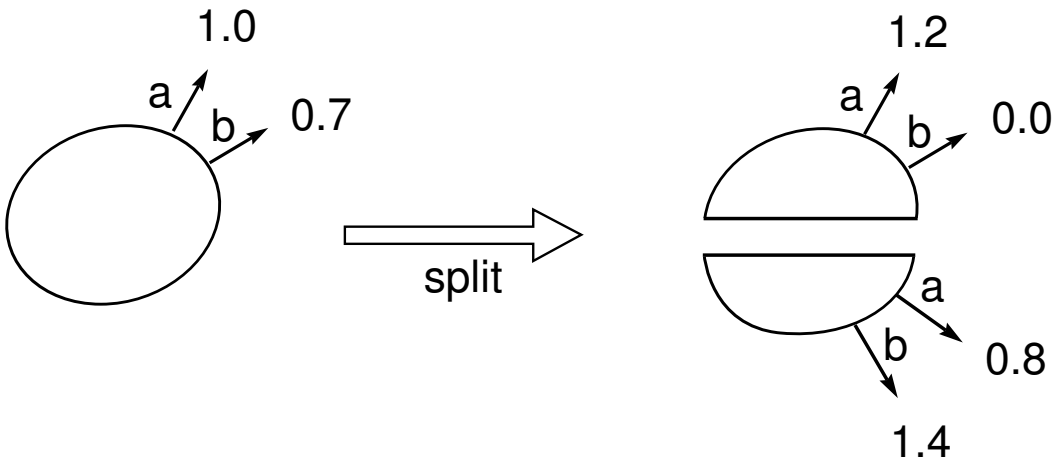


Figure 15: A necessary policy distinction: The state grouping must be split to avoid mistakes in policy.

make *sound decisions* in the task, in terms of a new property I call *incremental regret.* It explains how failing to make policy distinctions or value distinctions may result in large incremental regret. The section defines the ϵ-*adequacy* criterion for the adequacy of the representation at a particular state, and develops state-compatibility criteria which specify when to separate states to create an ϵ-adequate partition representation. These ideas formalize the 'necessary distinctions' aspect of cognitive economy into criteria which may be implemented in on-line algorithms.

3.5.1 Policy distinctions

Figure 15 illustrates a distinction which must be made by the representation for the agent to make correct decisions in its task. The figure shows the original generalized state (on the left), and a pair of generalized states (on the right) which result when it is split. In the split region, action a is much better than action b for the group of states on the top, while this preference is reversed for the states in the bottom group. The original generalized state is a mixed set of states; because its action values are

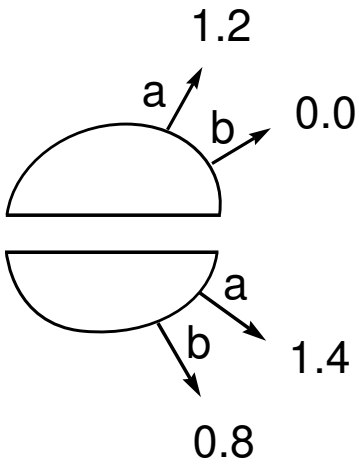


Figure 16: An *un*necessary distinction: Splitting the generalized state is probably not worth-while.

averaged over all its members, it slightly prefers action a, which is the wrong policy for the states in the bottom of the region. Failure to make this policy distinction causes the agent to make bad decisions for the bottom states. In contrast, Figure 16 shows a split region where the top and bottom halves both prefer action a; we may simplify matters by grouping them together in a single generalized state.

3.5.2 Value distinctions

If states require the same behavior, their differences can often be ignored—but not always. Figure 17 shows generalized state S_1, from which action a leads to S_2, having action values $(-1.0, 0.0)$, and action b leads to S_3, which has action values $(0.0, 1.0)$. In this case, the separation between the generalized states S_2 and S_3 is necessary even though S_2 and S_3 have the same action preference (take action b). A representation which combined S_2 and S_3 could support a correct policy, but would not allow the agent the means to *learn* one from its experience because the agent would seemingly arrive at the same state by taking either action from S_1. The two outcomes must be

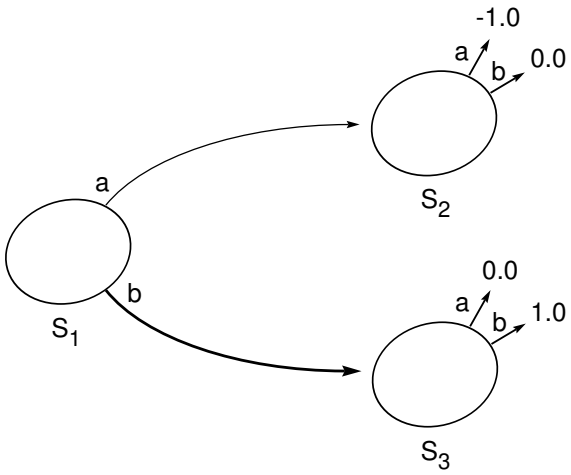


Figure 17: A necessary value distinction: The resulting states must be kept separate for the agent to seek the better outcome.

separated if the agent is to be able to learn to choose action b from S_1, in order to make possible an expected reward of 1.0 (in S_3) instead of 0.0 (in S_2).

Value distinctions serve the same purpose as policy distinctions: they enable the agent to learn to make sound decisions in its task. The distinctions between states with different preferred policies are *forward-looking,* allowing the agent to choose its *next* action wisely. The distinctions between states with different values (but possibly the same preferred policies) are *backward-looking,* enabling the agent to distinguish different outcomes for the actions it selects from a *previous* state. Both forward and backward distinctions are necessary in allowing the agent to learn to make sound decisions. Figure 18 illustrates the forward and backward-looking effects of state generalization at state s_t.

3.5.3 Making sound decisions

We have seen that the representation must avoid generalizing over certain states if the agent is to be able to solve its task. When the representation fails to make necessary

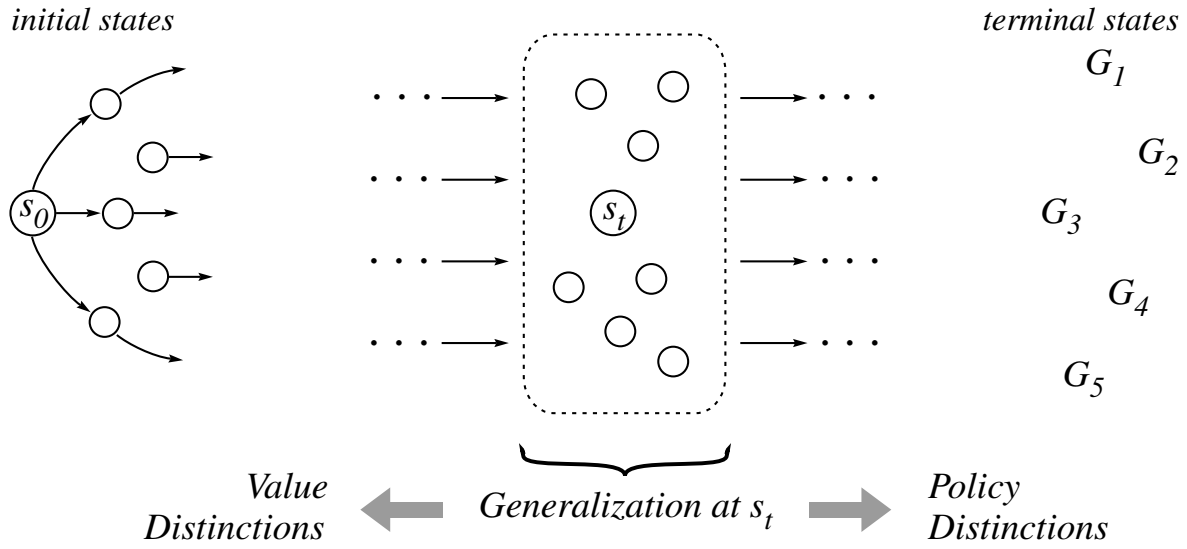


Figure 18: The representation must make appropriate policy distinctions, which affect its next move, and value distinctions, which allow wise choices from earlier states.

distinctions, the agent is unable to choose actions which maximize its expectation of reward from the task (the accumulated reinforcement from the environment). If ignoring the distinction results in a small enough loss of potential reward, we may want to ignore the distinction, in favor of a simpler representation which enables the agent to learn the task more quickly and easily. How much lost reward is too much?

If we want the agent to always make perfect decisions, any loss of reward at all is too much. A more practical criterion is that the agent be able to choose actions which are close to optimal at each state. Therefore, we will define sound decisions as choices which are within some tolerance of the best actions. We should also take into account the information available to the agent. From the standpoint of an omniscient observer, the agent may appear to be making poor choices; but these may still be sound decisions, given the limited information available at that point in the task. We will say that the agent is making sound decisions if it chooses actions which appear best according to a limited look-ahead, given some tolerance for allowable error.

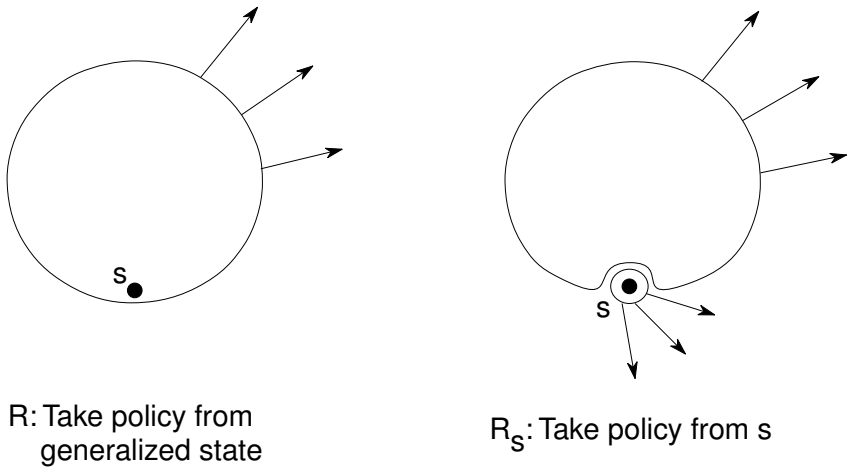


Figure 19: Incremental regret: Compare R, the expected long-term reward from s when we act according to the policy of the generalized state, with R_s, the return that results from taking the action which is best at s itself.

The general idea is shown by Figure 19: compare the reward the agent would see if it takes its policy from the generalized state with the reward it would see if it selects its action according to the action values for its current state, s. The difference between these two estimates is the error which would result from missing a necessary distinction at s. If this error is too large, we should make a distinction between s and the other members of the generalized state.

The regret of a representation

Ideally, we would like to be able to evaluate actions in terms of the total reward the agent would experience over the remainder of the task episode. If we could do this, we might define the *regret of a representation* as the expected loss of reward, due to the representation's failure to make necessary state distinctions. This measurement would indicate how much reward the agent stands to lose if it adopts a particular representation instead of the discrete representation (assuming that the action values

are learned correctly and the agent follows the greedy policy). Bad representations would thus be those with large regrets. This usage of the term regret is different than that usually found in the literature. In (Kaelbling, Littman and Moore, 1996) and (Berry and Fristedt, 1985), regret refers to a measure of the effectiveness of a particular *algorithm,* and may measure the performance over many task episodes. Here we are using the term to characterize the post-convergence performance of the system with different *representations.*

Incremental regret

Unfortunately, the regret of the representation depends upon complete episodes of the task, but we want to base our criterion for sound decisions on a limited look-ahead. We want to isolate the effect the representation has on the quality of individual decisions made by the agent. For example, if the current state is s, the value $Q(s, a_i)$ gives an estimate of the expected reward for choosing action a_i. If state generalization at s has resulted in compromises in the action values, $Q(s, a_i)$ may be averaged over a large number of states, but inaccurate at s. Suppose that action a_i would lead to state s'. To get a more accurate estimate of the value of a_i at s, we can look ahead one step, and consider the sum of the immediate reward for choosing a_i and the value of the resulting state, given by $\max_j Q(s', a_j)$. The action which is best according to this one-step look-ahead may be different from the action which maximizes $Q(s, a)$. We will call the difference in expected reward for these two actions the *incremental regret.* The incremental regret thus reflects the compromises in action values caused by state generalization at s : it measures the severity of missing a policy or value distinction. We will say that an adequate representation is one having low incremental regret at

each state; therefore, the action values will be accurate enough for the agent to make sound decisions—to always choose an action which is within some tolerance of the true best action, according to a limited look-ahead from the current state.

Low incremental regret contributes to low total regret, but this relationship is complicated by the discounting of future rewards. Since we define action values in terms of *discounted* future rewards, we cannot determine the contribution of the incremental regret of a particular state toward the total regret without knowing how far each reward lies in the future. For example, an agent may make a single bad move at its initial state, with a small incremental regret. But that small incremental regret could be related to a large total regret if the loss of reward occurs only at the end of a long episode. In this case, the eventual loss would be heavily discounted. To further complicate matters, the amount of discounting also depends on the agent's choice of action, since different actions may result in episodes of different lengths. Consequently, adjusting the representation to minimize incremental regret will not necessarily minimize total regret; however, incremental regret may be the appropriate measure of performance, given our assumptions regarding optimal behavior. Our model of optimality discounts reward by its distance into the future, and the agent makes its decisions on the basis of discounted reward without considering how far away that reward may be.

3.5.4 Criterion for representational adequacy

The previous paragraphs defined an adequate representation as one having low incremental regret at each state, allowing the agent to make sound decisions. The following paragraphs define the ϵ-adequacy criterion, which allows us to say whether the incremental regret is always less than ϵ, a predefined tolerance for the amount of error we

will tolerate before deciding that a policy or value distinction is a necessary one. The criterion considers whether the representation appreciably changes the agent's view of the preference set or overall value of its current state.

The apparent value of a state s is calculated from the generalized action values:

$$V(s) = Q(s, a_k) = \sum_i w_{ik} f_i(s)$$

where action a_k is the action which results in the greatest value of Q at s. If the representation groups s together with states that are incompatible with it, then the generalized action values w_{ij} may be compromised in ways which lead to inaccurate action values for s. These inaccurate action values may cause the agent to see an inaccurate value for $V(s)$ as a result, or may cause the agent to erroneously view some inferior action as preferable to a_k at s. In the first case, the agent may fail to make necessary value distinctions, and in the second case, the agent may fail to make necessary policy distinctions at s.

Therefore, we will require that the representation present s with enough accuracy that the preference set and overall state value are close to what they would be if there was no generalization over s and "neighboring" states. To do this, we calculate the action values for s individually by conducting a one-step look-ahead from s, in order to compare them with the values taken from the generalized state. Specifically, we will require that $\text{pref}_\delta(s)$ only contain actions which would be preferred according to the one-step look-ahead values, and that $V(s)$ be close to the value which we would get via the same one-step look-ahead function. By using a one-step look-ahead, we thus bypass the effects of state generalization at s. Hence, the first step in defining a criterion for representational accuracy will be to define a one-step look-ahead action value function, and one-step look-ahead versions of $V(s)$ and $\text{pref}_\epsilon(s)$.

$Q1(s,a)$—action value function with a one-step look-ahead

The action value function $Q(s,a)$ is subject to the effects of state generalization at s. That is, its value at s not only depends on s, but on the values at other states s' in the same generalized state. Therefore, we define another action value function, $Q1(s,a)$, which is the sum of the immediate reward and the value of the resulting state. The difference between $Q1$ and Q at s indicates the extent to which the representation's generalized action values are compromised at s.

Definition 3.1 ($Q1(s,a)$) *We define the function $Q1$ for the value of an action, based on the expected discounted future reward seen one step in the future. Let s be the state seen by the agent at time t, and s' a possible resulting state at time $t+1$. We define*

$$Q1(s,a) = \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$$

where

$$V(s') = \max_j Q(s', a_j) = \max_j \sum_i w_{ij} f_i(s')$$

Thus $Q1(s,a)$ represents the agent's best estimate of expected reward for taking action a from state s, given a one-step look-ahead. But the agent makes its decisions on the basis of

$$Q(s,a_j) = \sum_i w_{ij} f_i(s)$$

The difference is that $Q1(s,a_j)$ is based on the actual result of taking action a_j from s, while $Q(s,a_j)$ is based entirely on the generalized action values, w_{ij}. The w_{ij} reflect the compromises made by the representation; therefore, w_{ij} says something about the expected reward for the *set of states* detected by f_i. Suppose that the best action from s is a_1, but that the detectors which are active for s ($f_i(s) > 0$) are also activated for

states which all prefer action a_2. Then we would expect that the generalized action values favor a_2; in other words, $w_{i1} < w_{i2}$, and $Q(s, a_1) < Q(s, a_2)$. As a result, the agent would incorrectly choose action a_2 from s. But $Q1(s, a_1) > Q1(s, a_2)$, since $Q1$ only considers the expected reward for s alone. If $Q1(s, a_1) - Q1(s, a_2)$ is large enough, we would say that the agent fails to make a necessary policy distinction at s.

$V1(s)$—state value function, based on a one-step look-ahead

Chapter 2 defined the state value function $V(s)$ in terms of the action values, $Q(s, a)$. Replacing the values $Q(s, a)$ with $Q1(s, a)$ allows us to define the state value in terms of a one-step look-ahead:

Definition 3.2 ($V1(s)$) *Let s be a state, and $a_i \in \mathcal{A}(s)$ an action available to the agent from state s. Then we define*

$$V1(s) = \max_i Q1(s, a_i)$$

Thus $V1(s)$ represents the expected value of state s, based on a one-step look-ahead from s.

$\mathbf{pref1}_\epsilon(s)$—preference set for s, based on a one-step look-ahead

The chapter defined $\text{pref}_\epsilon(s)$ in terms of the action values, $Q(s, a)$. Replacing the values $Q(s, a)$ with $Q1(s, a)$ allows us to also define the preferred action set according to a one-step look-ahead:

Definition 3.3 ($\mathbf{pref1}_\epsilon(s)$) *Let s be a state, and $a_i \in \mathcal{A}(s)$ an action available to the agent from state s. Then we define*

$$\mathit{pref1}_\epsilon(s) = \{a \in \mathcal{A}(s) : Q1(s, a) \geq \max_i Q1(s, a_i) - \epsilon\}$$

Thus $\text{pref1}_\epsilon(s)$ is the set of "best actions" for the agent at s, according to a one-step look-ahead.

The criterion for representational adequacy

Having defined the preferred action set and state value according to a one-step look-ahead, we can easily compare these with the agent's information at s in order to evaluate the accuracy and adequacy of the representation at s :

Definition 3.4 (ϵ-adequacy) *Let δ be given, and assume that the agent always selects an action from $pref_\delta(s)$. We will say that a representation of the state-space is an ϵ-adequate representation for $\delta \leq \epsilon$, if for every state $s \in \mathcal{S}$ reachable by the agent, the following two properties hold:*

$$|V1(s) - V(s)| \leq \epsilon$$

$$pref_\delta(s) \subseteq pref1_\epsilon(s)$$

The meaning of ϵ-adequacy

Meeting the ϵ-adequacy criterion guarantees that the state generalization at s does not prevent the agent from being able to learn the correct policy at s or mislead the agent at an earlier state as to the desirability of s. Thus, the criterion defines a standard for representational accuracy at individual states, guaranteeing that the harmful effects of state generalization are kept in check and that the agent can learn to make sound decisions. This is our standard for an *adequate representation,* one which makes the distinctions needed for the task to remain learnable.

The ϵ-adequacy criterion is well-suited to our on-line learning problem because it is based on the regret associated with a single decision. The agent typically chooses

its actions on the basis of a one-step look-ahead; it does not have access to the true action values for its actions, but instead it sees *generalized action values* for the states immediately ahead, filtered by the representation. Basing the adequacy criterion on such a small window into the state-space—a single step—does mean that we ignore certain other considerations. For example, if a task contains cycles or repeated states, it is possible that the incremental regret of repeated states may affect the total regret more than the incremental regrets at other states; but the single-step criterion does not allow detection of cycles. Furthermore, the criterion is recursive, in the sense that the agent needs the criterion to hold for its future states before the agent can use it effectively in its current state. This is not a concern for a static representation, since we simply say that it meets the ϵ-adequacy criterion at all states. But if we are using the criterion to construct such a representation through on-line learning, the agent needs the representation to be adequate for the remainder of the episode in order for the agent to make correct value distinctions at its current state.

Such limitations are common to most algorithms for reinforcement learning, since the values of earlier states cannot be determined until the values of the later states stabilize. The effect is that of a "frontier," beyond which the action values and representation are completely learned, while the values of states in front of the frontier are still in flux. Initially, the frontier is at the final states of the task episodes. As the agent learns, and correct information percolates backwards from the ends of episodes to the beginning states, this frontier moves backwards toward the start of the episode, arriving at the initial state as the task is learned. The ϵ-adequacy criterion allows the agent to consider each decision separately, but is only informative at the frontier of learning—as are the other mechanisms for reinforcement learning, such as Q-learning.

Although the ϵ-adequacy criterion provides an objective standard for an adequate representation—one which allows the agent to learn its task—these characteristics of the representation are really the outcome of the particular distinctions the representation makes or fails to make between individual states. Thus we need to bridge the gap between the high-level description of adequate representations and the low-level decisions the agent must make as to which states must be kept separate. In other words, when must the representation distinguish states and when may it generalize over states, in order for it to be ϵ-adequate? The next section addresses this issue.

3.5.5 State Compatibility

We want a set of low-level criteria which specify when states may be grouped together, and when they must be represented separately. One approach is to compare the information we have for the current state with the information for the group of states it belongs to. For example, we could compare $V1(s)$ directly with $V(s)$, and we could compare $\text{pref1}_\epsilon(s)$ with $\text{pref}_\delta(s)$. In this way, we may evaluate the ϵ-adequacy of the representation at s, and make additional distinctions if this criterion does not hold. This approach is reminiscent of Carpenter and Grossberg's ART (Carpenter and Grossberg, 1988), in which the current observation results in a new category if its best classification is a poor match. Thus the ϵ-adequacy criterion essentially defines the compatibility of states with the regions to which they belong.

This section presents criteria for the compatibility of one state with another. State-state compatibility may be more useful in on-line feature extraction than state-region compatibility because a poorly-chosen region could be incompatible with *all* its member states. Comparing individual states removes the effect of state generalization at the

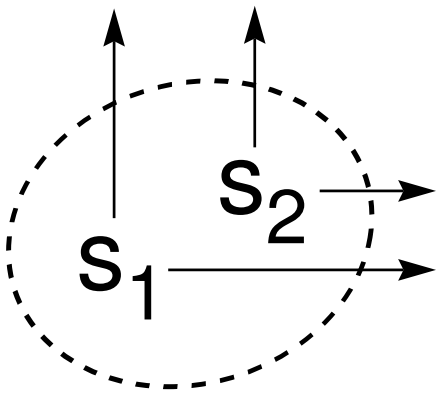


Figure 20: State compatibility: Allow s_1 and s_2 to be grouped together if a one-step look-ahead reveals their overall values to be close and their preference sets similar.

current point in the task episode, and allows access to the more reliable information gained by looking ahead one step. Figure 20 illustrates the general strategy, which is to look ahead one step from each of a pair of states, so that we may consider whether grouping them obscures any necessary distinctions.

An agent which attempts to choose the best action may actually choose any of the actions in its preference set. We will assume that δ is the tolerance which defines those sets, and is already determined; thus the agent might select any action in $\text{pref}_\delta(s)$. Our goal is to make sure that this choice results in an incremental regret of no more than ϵ. If the representation made no compromises in the action values, then this would always be the case for $\delta \leq \epsilon$. More typically, the representation makes state generalizations which cause deviations from the true action values. The purpose of the compatibility criteria is to make sure that these errors are not serious enough to prevent the representation from being ϵ-adequate.

Compatibility criteria

Definition 3.5 (State compatibility) *Let s_1 and s_2 be states in $\mathcal{S}$. Let δ be given, and assume that the agent always selects an action from $pref_\delta(s)$. Assume that our goal is to produce an ϵ-adequate representation, where $\epsilon \geq \delta$.*

We will say that s_1 and s_2 are compatible *in case the following three conditions hold:*

$$pref1_\epsilon(s_1) = pref1_\epsilon(s_2) \tag{19}$$

$$|V1(s_1) - V1(s_2)| \leq \delta \tag{20}$$

and

$$pref1_\delta(s_1) = pref1_\delta(s_2) \quad \text{if } \delta \leq \epsilon/2 \tag{21}$$

$$pref1_0(s_1) = pref1_0(s_2) \quad \text{otherwise} \tag{22}$$

The criteria consist of three rules. The first rule ensures that the same actions appear desirable in each state. The second rule requires that the values of the states are close, based on a one-step look-ahead. The purpose of the third rule (Equations 21 and 22) is to ensure that the action which appears to be the best for a set of compatible states is, in fact, a pretty good action for any of the states in the set. This is difficult to guarantee when the compatibility criteria are written for pairs of states, rather than in terms of the whole set. That is why the criteria demand equality of the preference sets instead of merely requiring the preference sets to overlap. When $\delta \leq \epsilon/2$, the looser restriction of Equation 21 allows the states to have slightly different values for the top actions, making the criteria more suitable for a practical algorithm which must account for real-world noise in the value estimates. The cut-off value of $\epsilon/2$ appears to come from the sum of the errors allowed by combining Equation 20 with Equation 21.

Regarding the compatibility of a set of states (for example, a partition region), we will say that all the states in the set are compatible if each pair of states is compatible. We will refer to such sets as *pure sets.*

Although these are certainly not the only possible criteria for state compatibility, I have not been able to find less restrictive criteria that meet my requirements and still guarantee ϵ-adequacy. I leave that to future work. The two following chapters discuss some of the theoretical issues in greater depth: Chapter 4 examines the role of generalized action values and shows how they may be defined as the "true," steady-state values which characterize a particular representation. Chapter 5 discusses the state compatibility criteria in more detail: it gives examples of what can go wrong when the compatibility criteria are violated, and it proves that for partition representations, separating incompatible states guarantees ϵ-adequacy of the representation.

3.6 Feature Extraction

The criteria developed in this chapter characterize representations that exhibit high cognitive economy. This is the "what" of representation; the process of feature extraction is the "how."

The feature-extraction problem is both heuristic and difficult. Learning the representation on-line requires the agent to learn the action values for its categories while it is reorganizing those categories to better describe its world. This problem combines the difficulties of action-value learning, state compatibility assessment, and data clustering, in order to group states into categories which best present the relevant aspects of the agent's world. Each of these components is a difficult problem by itself. In the general case, a good algorithm for on-line feature extraction must be able to propose trial

distinctions between states, test them for relevance in the task, and possibly discard them. Chapter 6 presents an algorithm which implements the criteria developed here, and which proved successful in two reinforcement learning tasks.

3.7 Summary

Wisdom consists in knowing which details matter, and when we may safely ignore them. To make wise choices, we must look to the requirements of the task, and categorize the elements of our world in ways which allow us to make sound decisions.

This chapter has reviewed the principle of cognitive economy, and has formalized it into objective criteria for representational adequacy and state compatibility. These characterize necessary distinctions in terms of generalized action values, and are therefore readily implemented in on-line algorithms for reinforcement learning tasks. The criteria are principled because they are tied to the requirements of an agent learning to choose actions which maximize its expected reward in its task.

Chapter 4

Action Values for Generalized States

> Make everything as simple as possible, but not simpler.
>
> —*Albert Einstein*

4.1 Introduction

The previous chapter formalized the principle of cognitive economy in terms of converged action values—the steady-state values which describe the agent's true expectation of reward, given a particular representation for its task. Although that chapter made use of action values for generalized states, it deferred their definition to this chapter.

For a discrete representation, action values are defined for individual state-action pairs; they estimate the agent's future experience of reinforcements as it proceeds from a particular state. This chapter extends the idea of action value by defining a generalized action value for the set of states recognized by a particular feature detector. To do this, the chapter considers a simple gridworld task with different representations: the discrete representation, a partition representation, and finally, representations with overlapping feature detectors. There are several ways to generalize the idea of action value to groups

of states; the chapter presents two methods of defining these values, each based on a different set of assumptions about the problem. The generalized action values enable us to see how the state generalization given by the agent's feature detectors distorts the agent's perception of the world by altering the apparent values of its actions in the task.

4.1.1 Action values depend on the agent's representation

If the agent has a discrete representation of its world, it distinguishes each combination of details as a unique state, and maintains separate action values for each of these states. If, instead, the agent's representation of the world generalizes over states, its action values may be different from the action values for the individual states. Thus the agent's perceived world may be one in which its action values and its optimal policy differ from that of the real world. To understand how this happens, we must understand how state generalization affects the agent's action values.

A simple gridworld task will motivate our discussion of state generalization. First, we will derive the usual state-action values for a discrete representation of the grid. Then we will extend the analysis to include region-action values for a partition representation of the grid. In order to further extend the analysis to representational schemes with overlapping features, we will need to make some additional assumptions. We will examine two approaches to extending the analysis in order to produce a definition of generalized action values which covers all these representational schemes.

4.2 Example: State-Action Values for a Discrete Representation

Consider again the two-action gridworld task, where the grid has four cells on a side, the starting state is in the lower left corner at (1, 1), the goal state is in the upper right corner at (4, 4), and the allowable actions are up and right. Figure 21a shows this gridworld with a discrete representation of the states. Since we cannot move down or

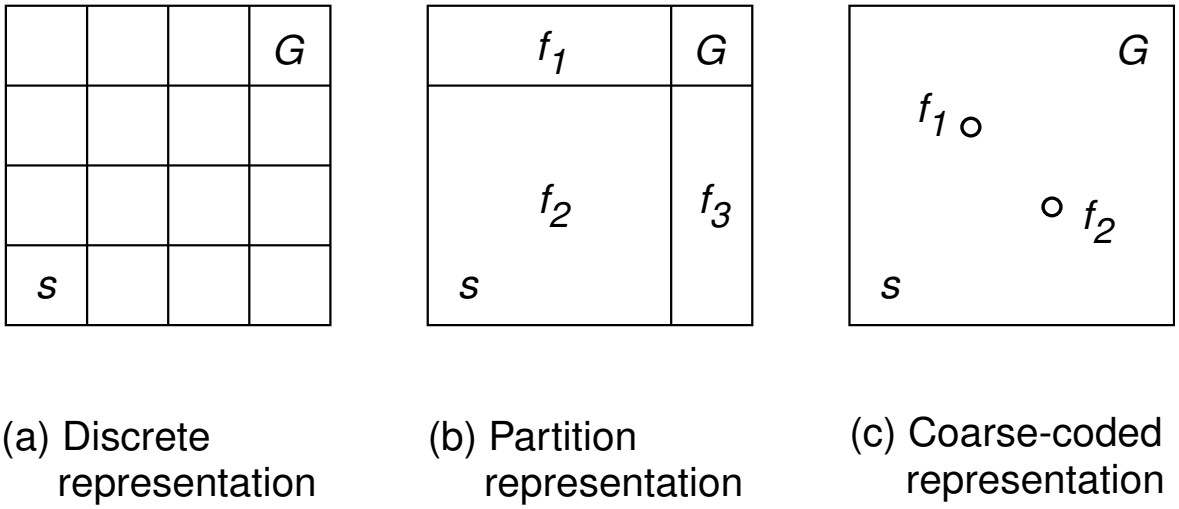


Figure 21: Three representations of the 4×4 gridworld

to the left, we cannot back-track or cycle between states. This makes the action values easy to calculate in the case of a discrete representation. The following discussion will use the notation $Q(s, a)$ for the action values, without an index for the time; the reason is that we are determining the final, stable set of action values. We can calculate the action values for an optimal policy as follows. First, we observe that moving up from the top row or right from the right edge results in a reward of -1, indicating failure. Moving right from cell $(3, 4)$ or up from $(4, 3)$ will result in the goal state and a reward of 1. All other actions simply move us one step closer to the end of an episode,

without presenting any immediate reward. Therefore, the value for each of these non-terminal actions is simply the final reward for the episode (which is 1 under an optimal policy), discounted according to the number of steps needed to reach the goal. Thus $Q((2,4),\mathsf{right}) = Q((3,3),\mathsf{right}) = Q((3,3),\mathsf{up}) = Q((4,2),\mathsf{up}) = \gamma$, because each of these actions sets the stage for us to move to the goal state in the next step. For the states which occur earlier, we will have $Q((x,y),a_i) = \gamma^{k-1}$, where k is the number of actions required to reach the goal from that state. Setting the discount factor $\gamma = 0.9$, the values for the action up are given by Table 3 and the values for the action right are given by Table 4.

-1	-1	-1	—
0.729	0.810	0.900	1
0.656	0.729	0.810	0.900
0.590	0.656	0.729	0.810

Table 3: Action values for up—discrete state representation

0.810	0.900	1	—
0.729	0.810	0.900	-1
0.656	0.729	0.810	-1
0.590	0.656	0.729	-1

Table 4: Action values for right—discrete state representation

So we have computed the action values, which represent the expectation of reward for individual states. Next, we will consider a coarse partitioning of the state-space and see how this changes the action values. Instead of considering the values $Q(s,a)$ for individual states s, we will calculate the expected reward for *regions* of states, since our new representation will not allow us to store distinct action values for the states within a partitioned region. We will find that the expected reward depends on our definition of the observed values, $< s_t, a_k >$. To avoid confusion, we will write $v_0^\pi(S_i,a)$ for the

expected reward if the agent takes action a from region $S_i \subseteq \mathcal{S}$ and then proceeds according to the policy π, with the subscript 0 indicating the one-step assumption regarding the observed values, $< s_t, a_k >$. (Recall from Chapter 3 that this means looking ahead one step, to the values of the next state—rather than looking ahead all the way to the end of a task episode). We will write $\upsilon_1^\pi(S_i, a)$ for the expected reward under the whole-path assumption regarding observed values. Thus υ_0 corresponds to TD(0); υ_1 corresponds to TD(1), which equates the value with the whole-path return. Before considering an actual example, let us first work out the theory.

4.3 Region-Action Values

Assume that the state-space is partitioned into a set of disjoint regions, S_i, which cover the space: $\bigcup S_i = \mathcal{S}$ and $S_i \cap S_j = \emptyset$, for $i \neq j$. For the moment, let us ignore the distinction between υ_0 and υ_1 and define the *region-action value* function according to

$$\upsilon^\pi(S_i, a) = E_\pi\{< s_t, a >\}$$

This means that the action value for the region S_i is the expected value of the observations for states $s_t \in S_i$, assuming the sampling policy π. Although algorithms like Q-learning may learn through *off-policy* learning, in which the sampling policy can be different from the policy we are evaluating, we will defer consideration of off-policy learning to the next section.

Since the representation cannot distinguish values for individual states in S_i, we express the expected return in terms of the conditional probabilities of the states $s \in S_i$:

$$\upsilon^\pi(S_i, a) = \sum_s \Pr\{s|S_i\}\upsilon^\pi(s, a) \tag{23}$$

From our previous analysis, we know that $\upsilon^{\pi}(s,a)$ is the expectation of the immediate reward for taking action a from s, plus the value of the following state:

$$\upsilon^{\pi}(s,a) = \sum_{s'} \mathcal{P}^{a}_{ss'} \left(\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s') \right) \tag{24}$$

As in Chapter 2, $\mathcal{P}^{a}_{ss'}$ represents the probability of arriving in state s' as the result of taking action a from state s, and $\mathcal{R}^{a}_{ss'}$ is the expectation of immediate reward for that transition.

The one-step and whole-path approaches differ in their estimate of the value of the remainder of the episode, $V^{\pi}(s')$. Under the whole-path assumption, this value is just the expectation, under the policy π, of the return from state $s' = s_{t+1}$:

$$V^{\pi}_{1}(s') = E_{\pi}\{R_{t+1}\}$$

(With off-policy learning, this expression becomes more complicated because we need to weight the returns by their relative probabilities under the evaluation policy and the sampling policy. Sutton and Barto (1998, pp. 124-129) discuss this in detail).

Under the one-step look-ahead assumption, we must define $V^{\pi}(s')$ in terms of our system's values at time $t+1$. Therefore

$$V^{\pi}_{0}(s') = \sum_{a'} \pi(S_{s'}, a') \upsilon^{\pi}_{0}(S_{s'}, a')$$

where $S_{s'}$ is the partition region which covers state s'. Substituting these state values back into Equation 24 yields the following pair of equations:

$$\upsilon^{\pi}_{1}(s,a) = \sum_{s'} \mathcal{P}^{a}_{ss'} \left(\mathcal{R}^{a}_{ss'} + \gamma E_{\pi}\{R_{t+1}\} \right) \tag{25}$$

$$\upsilon^{\pi}_{0}(s,a) = \sum_{s'} \mathcal{P}^{a}_{ss'} \left(\mathcal{R}^{a}_{ss'} + \gamma \sum_{a'} \pi(S_{s'}, a') \upsilon^{\pi}_{0}(S_{s'}, a') \right) \tag{26}$$

Thus we can find the values for each state-action pair (s,a) either by means of the actual returns under a particular policy, or by solving a system of equations involving

the one-step look-ahead estimates. Together with Equation 23, this gives us a set of action values for the partitioned regions.

We can make some general observations about how the representation affects the action values, based on Equations 25 and 26. Under the whole-path approach, the action values $\upsilon_1^\pi(s,a)$ will agree with those of a discrete representation, (or any other representation, for that matter) as long as we use the same policy, π. The one-step approach will usually differ from the discrete representation on the action values $\upsilon_0^\pi(s,a)$, because it defines $\upsilon_0^\pi(s,a)$ in terms of aggregate values $\upsilon_0^\pi(S_j,a)$, instead of directly in terms of the actual rewards that the agent will experience. With either approach, the *region*-action values $\upsilon^\pi(S_i,a)$ will usually differ from the corresponding values for a discrete representation because of the way Equation 23 averages the values over the states $s \in S_i$.

Note that our action values are defined in terms of a particular policy, because the policy determines our path and the resulting reward after the first action. It is important to remember that our choice of representation determines which policies are valid. For a policy to be valid, it must not assign different actions to states which are grouped together in the same region. Hence these equations for the action values only make sense if our policy, π, does not discriminate between states which lie in the same region, but assigns them the same set of action probabilities. Essentially, we have collapsed each region into a generalized state, which has a single value for each action and a single policy.

A second reason why we must specify the policy is that the system dynamics depend on how we sample the states, if we have state generalization. The expected value of taking action a in region S_i depends on which state in S_i is our starting point.

With state generalization, changing the policy, π, changes the conditional probabilities $\Pr\{s|S_i\}$, and thus changes $v^\pi(S_i, a)$ (Equation 23).

Normally we will want to know the action values for an optimal policy—the optimal action values—but we must bear in mind that the best values under a particular representation could be worse than the optimal action values under a discrete representation. One reason is that when we group states together into regions, we end up averaging their action values, possibly "watering down" the optimal values. A second reason is that the policy which is optimal under the discrete representation may be invalid for the particular representation under investigation. In that case the best policy which our representation can support may lead to smaller rewards than we could achieve with a discrete representation.

Let π^* denote an optimal policy for our current representation, and let v^* denote the expected reward function under π^*. Because π^* is optimal, it only takes non-zero values for actions a' which maximize $v^*(S_i, a')$. Therefore, we may simplify Equation 26 for the one-step values:

$$v_0^*(s, a) = \sum_{s'} \mathcal{P}_{ss'}^a \left(\mathcal{R}_{ss'}^a + \gamma \max_{a'} v_0^*(S_{s'}, a') \right) \tag{27}$$

If we substitute this expression for $v_0^*(s, a)$ in Equation 23, the result is a generalized Bellman optimality equation for v_0^* :

$$v_0^*(S_i, a) = \sum_s \Pr\{s|S_i\} \sum_{s'} \mathcal{P}_{ss'}^a \left(\mathcal{R}_{ss'}^a + \gamma \max_{a'} v_0^*(S_{s'}, a') \right) \tag{28}$$

The Bellman equation is a way of writing the constraints which must hold between the action values at times t and $t + 1$ if we accept the one-step assumption.

4.3.1 Example: Region-action values for a partition representation

To illustrate these ideas, suppose we have the partition representation of the gridworld given by Figure 21b, and we want to calculate the action values under an optimal policy. We will simplify matters for this example by choosing a deterministic, optimal policy. In this representation, we have three detector regions: $\mathcal{S}_{f_1}$ covers the top row except for the goal state, $\mathcal{S}_{f_3}$ covers the right edge, and $\mathcal{S}_{f_2}$ covers the interior states. In the top row, moving up is fatal; therefore the optimal policy action in $\mathcal{S}_{f_1}$ is right. Since the action right is fatal in $\mathcal{S}_{f_3}$ the optimal action for $\mathcal{S}_{f_3}$ is up. The policy for the interior does not matter, so we may arbitrarily choose the action for $\mathcal{S}_{f_2}$ to be up. Thus we have an optimal policy, resulting in the following path through the state space: $(1,1),(1,2),(1,3),(1,4),(2,4),(3,4),(4,4)$, with a reward of 1 on the last step. (See Figure 22).

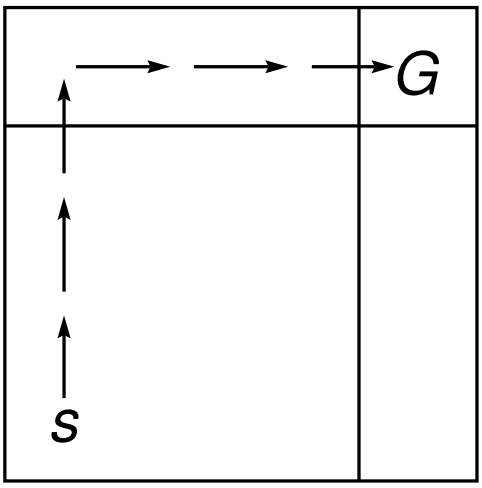


Figure 22: An optimal path through the partitioned gridworld

Calculating v_1^π — whole-path assumption

If we make the assumption that the values depend on the return for the whole path, we look at the states $s \in S_i$, calculate the action values for (s, a) according to Equation 25, and substitute them in Equation 23 to arrive at $v_1^*(S_i, a)$ for each of the three regions S_i. We can simplify the application of Equation 25 by noting that for each allowable state transition, $\mathcal{P}^a_{ss'} \equiv 1$, since the policy is deterministic. Since the reward $\mathcal{R}^a_{ss'} = 0$ except for the last transition, where the reward is 1, we find that $v_1^*(s, a) = \gamma^{k-1}$, where k is the number of actions needed to reach the goal from s. These are the same values of $v^*(s, a)$ that we calculated for the discrete representation, shown in Tables 3 and 4.

Consider detector f_1, which covers the top row (except for the goal state). Here $v_1^*(\mathcal{S}_{f_1}, \mathsf{up}) = -1$, since moving up from the top row is always fatal. For the action right, we plug the values $v_1^*(s, a)$ into Equation 23, taking $\Pr\{s|S_i\} = \frac{1}{3}$, since we visit each of the three states in $\mathcal{S}_{f_1}$ with the same frequency under our policy. Thus

$$\begin{aligned} v_1^*(\mathcal{S}_{f_1}, \mathsf{right}) &= \frac{1}{3}\left(v_1^*((1,4), \mathsf{right}) + v_1^*((2,4), \mathsf{right}) + v_1^*((3,4), \mathsf{right})\right) \\ &= \frac{1}{3}(\gamma^2 + \gamma + 1) \\ &\doteq 0.903 \end{aligned}$$

Detector f_2 will only be active for states $(1,1), (1,2)$ and $(1,3)$ under our policy. Therefore,

$$\begin{aligned} v_1^*(\mathcal{S}_{f_2}, \mathsf{up}) &= \frac{1}{3}\left(v_1^*((1,1), \mathsf{up}) + v_1^*((1,2), \mathsf{up}) + v_1^*((1,3), \mathsf{up})\right) \\ &= \frac{1}{3}(\gamma^5 + \gamma^4 + \gamma^3) \\ &\doteq 0.659 \end{aligned}$$

To calculate $v_1^*(\mathcal{S}_{f_2}, \mathsf{right})$, note that if we move right from any of the cells covered by $\mathcal{S}_{f_2}$, we are one step closer to the goal, just as if we had moved up. Hence the expected

return, $E_\pi\{R_{t+1}\}$, is the same whether we moved up or right; therefore, $v_1^*(\mathcal{S}_{f_2}, \mathsf{right}) = v_1^*(\mathcal{S}_{f_2}, \mathsf{up}) \doteq 0.659$.

Finally, consider detector f_3, which covers the right edge of the grid. Equation 25 results in state-action values $v_1^*(s,a)$ which mirror those of the top row, except that the actions are reversed. (From the right edge, right is fatal, but up leads to the goal). But Equation 23 requires $\Pr\{s|\mathcal{S}_{f_3}\}$, even though the region $\mathcal{S}_{f_3}$ is not visited under our policy. Therefore, we may wish to define this conditional probability to be zero, in which case we get zeros for the values of $\mathcal{S}_{f_3}$ If, instead, we take the probability of the states as equally likely (in the non-event of visiting $\mathcal{S}_{f_3}$), we find that

$$\begin{aligned} v_1^*(\mathcal{S}_{f_3}, \mathsf{up}) &= \frac{1}{3}\left(v_1^*((4,1), \mathsf{up}) + v_1^*((4,2), \mathsf{up}) + v_1^*((4,3), \mathsf{up})\right) \\ &= \frac{1}{3}(\gamma^2 + \gamma + 1) \\ &\doteq 0.903 \end{aligned}$$

and $v_1^*(\mathcal{S}_{f_3}, \mathsf{right}) = -1$. Tables 5 and Table 6 summarize the action values for our partitioned representation, under the whole-path assumption regarding returns.

-1	—
0.659	0.903

Table 5: Whole path values for up—partition representation

0.903	—
0.659	-1

Table 6: Whole path values for right—partition representation

Note that the new action values differ from those of the discrete representation, even though the values $v_1^*(s,a)$ for individual states agree with those of the discrete

representation. Under the whole-path assumption, the representation does not change the rewards which the agent experiences, but it does change the agent's expectation of reward because of the way the action values are averaged over the states in a region (Equation 23).

Calculating v_0^* — one-step assumption

If we make the assumption that action values are given by a one-step look-ahead, we can proceed as follows. First, consider detector f_1. Since moving up from the top row is fatal, as we have already observed, $v_0^*(\mathcal{S}_{f_1}, \mathsf{up}) = -1$. To find the value of moving right in the top row, we must solve Equations 23 and 27 for these states. Since the top detector covers three states, and they are visited with equal frequency, Equation 23 yields

$$v_0^*(\mathcal{S}_{f_1}, \mathsf{right}) = \frac{1}{3}\left(v_0^*((1,4), \mathsf{right}) + v_0^*((2,4), \mathsf{right}) + v_0^*((3,4), \mathsf{right})\right) \qquad (29)$$

To apply Equation 27 to the states in $\mathcal{S}_{f_1}$, recall that $\mathcal{P}^a_{ss'} \equiv 1$, since the task is deterministic, and $\mathcal{R}^a_{ss'} = 0$ except for the last transition, where it equals 1. Since moving right is preferable to moving up for any of the states in $\mathcal{S}_{f_1}$, we may replace $\max_{a'} v_0^*(S_{s'}, a')$ with $v_0^*(\mathcal{S}_{f_1}, \mathsf{right})$ in Equation 27 (except, of course, for the last transition, which takes us out of $\mathcal{S}_{f_1}$ and leaves us in the absorbing goal state, G). Therefore the values $v_0^*(s,a)$ we need to substitute in Equation 29 are as follows:

$$\begin{aligned} v_0^*((1,4), \mathsf{right}) &= 0 + \gamma v_0^*(\mathcal{S}_{f_1}, \mathsf{right}) \\ v_0^*((2,4), \mathsf{right}) &= 0 + \gamma v_0^*(\mathcal{S}_{f_1}, \mathsf{right}) \\ v_0^*((3,4), \mathsf{right}) &= 1 \end{aligned}$$

The solution is

$$v_0^*(\mathcal{S}_{f_1}, \mathsf{right}) = \frac{1}{3}\left(2\gamma v_0^*(\mathcal{S}_{f_1}, \mathsf{right}) + 1\right)$$

resulting in

$$v_0^*(\mathcal{S}_{f_1}, \mathsf{right}) = \frac{1}{3 - 2\gamma} \doteq 0.833$$

We compute the values for the interior detector in the same way. When we calculate the one-step look-ahead, we will need to know which actions yield the maximum values for $\mathcal{S}_{f_1}$ and $\mathcal{S}_{f_2}$. We have already seen that the best action for the top row is right. We will find that the best action for $\mathcal{S}_{f_2}$ is up, by the following reasoning. The representation groups the states in the top row into a single generalized state. If we move right from a cell on the left edge of the grid and then resume our policy, we incur an additional discount factor for that step, and we enter the top row at a point which is one cell closer to the goal. But the look-ahead value for the top row is the same, no matter what our entry point; we see $\max_a v_0^*(\mathcal{S}_{f_1}, a) = v_0^*(\mathcal{S}_{f_1}, \mathsf{right}) \doteq 0.833$. Therefore, moving right from $(1,1), (1,2)$, or $(1,3)$ will appear to be a wasted move, resulting in a smaller expected reward than immediately moving up.

To find $v_0^*(\mathcal{S}_{f_2}, \mathsf{up})$ we solve the following set of equations:

$$v_0^*(\mathcal{S}_{f_2}, \mathsf{up}) = \frac{1}{3}\left(v_0^*((1,1), \mathsf{up}) + v_0^*((1,2), \mathsf{up}) + v_0^*((1,3), \mathsf{up})\right)$$

$$\begin{aligned} v_0^*((1,1), \mathsf{up}) &= 0 + \gamma v_0^*(\mathcal{S}_{f_2}, \mathsf{up}) \\ v_0^*((1,2), \mathsf{up}) &= 0 + \gamma v_0^*(\mathcal{S}_{f_2}, \mathsf{up}) \\ v_0^*((1,3), \mathsf{up}) &= 0 + \gamma v_0^*(\mathcal{S}_{f_1}, \mathsf{right}) \end{aligned}$$

Thus we find that

$$v_0^*(\mathcal{S}_{f_2}, \mathsf{up}) = \frac{1}{3}\left(2\gamma v_0^*(\mathcal{S}_{f_2}, \mathsf{up}) + \gamma v_0^*(\mathcal{S}_{f_1}, \mathsf{right})\right)$$

Collecting the terms for $v_0^*(\mathcal{S}_{f_2}, \mathsf{up})$,

$$v_0^*(\mathcal{S}_{f_2}, \mathsf{up}) = \frac{\gamma v_0^*(\mathcal{S}_{f_1}, \mathsf{right})}{3 - 2\gamma}$$

Applying our previous result, this yields

$$v_0^*(\mathcal{S}_{f_2}, \mathsf{up}) = \frac{\gamma(\frac{1}{3-2\gamma})}{3 - 2\gamma} = 0.625$$

By our previous reasoning, $v_0^*(\mathcal{S}_{f_2}, \mathsf{right})$ is simply the value for moving up, but discounted once for the "wasted" step. Thus

$$v_0^*(\mathcal{S}_{f_2}, \mathsf{right}) = \gamma v_0^*(\mathcal{S}_{f_2}, \mathsf{up}) \doteq 0.563$$

Finally, we note that the values for $\mathcal{S}_{f_3}$ would theoretically be the same as those for $\mathcal{S}_{f_1}$, but reversed. Tables 7 and 8 summarize the results for our partitioned representation under the one-step assumption.

-1	—
0.625	0.833

Table 7: One-step values for up—partition representation

0.833	—
0.563	-1

Table 8: One-step values for right—partition representation

Note that the one-step action values are different from the values we computed under the whole-path assumption, and that the action values for up and right are no longer equal in the interior. Since the Bellman equations are based on the one-step look-ahead assumption, this example shows that the whole-path values will not necessarily

satisfy the Bellman equations under state generalization. Thus the use of Monte Carlo methods with value-function approximation will usually violate the Bellman equations.

4.4 Generalized Action Values

We are extending the definition of action value to representations which generalize over states. We call these values *generalized action values.* In the previous example, our representation was a partition, which divided the state-space into non-overlapping regions. At any given time, a single detector was active and unambiguously indicated which region contained the current state. Therefore, we could apply our knowledge of the individual states in the active region (their probabilities and their expected returns) to compute the action values.

Unfortunately, this approach breaks down in the general case, where the system state is described by a pattern of activity across multiple feature detectors, and the detectors may be continuous-valued functions. The equations we have developed, beginning with Equation 23, do not account for the influence of multiple detectors in our assessment of the current state. Yet it is the collective activity of the detectors which indicates the current state, and in our system model, determines the current value estimate, $Q^\pi(s, a)$. Therefore, we should not assume that there is a single region, $S_{s'}$ which is active after the transition from state s to state s' in Equation 26. We must also extend the analysis to consider the level of activation of the detectors; so far we have simply added in the values for the states covered by the active region without allowing for the possibility that the detector may respond more strongly to some states than others.

It is difficult to proceed without invoking some additional assumptions. I will present

two methods of defining generalized action values, each based on a different set of assumptions about the problem. The first method makes assumptions about the feature detectors, allowing us again to calculate the probabilities of individual states. The second method relates the action values to the assumptions inherent in a minimization of the system error. In both cases, we are simply defining what we mean by an action value in the general case, so that we can go on to examine how state generalization affects these values, and how the resulting changes in the action values affect the cognitive economy of the representation.

4.4.1 Method 1: Exploit assumptions of soft state aggregation

Suppose that each detector is associated with a particular cluster of states, and that the output of the detector represents the probability that the current state belongs to that cluster. That is, for a detector f_i associated with a cluster of states S_i, if the current state is s then $f_i(s) = \Pr\{S_i|s\}$. If we allow these probabilities to be between 0 and 1, we have *soft clustering,* in which there is still a single active region, but its identity is only known stochastically. Singh, Jaakkola and Jordan (1995) take this approach in their paper on *soft state aggregation.*

For the one-step approach, we proceed as follows. Making use of Equation 23 for $\upsilon^\pi(S_i, a)$, we have

$$\upsilon_0^\pi(S_i, a) = \sum_s \Pr\{s|S_i\}\upsilon_0^\pi(s, a) \tag{30}$$

This equation requires $\Pr\{s|S_i\}$. Fortunately, we can use Bayes' Rule to define this conditional probability in terms of the detector values $f_i(s) = \Pr\{S_i|s\}$ and the steady-state probabilities of states s' under our policy π (which we will write as $P_\pi(s')$).

Therefore

$$\Pr\{s|S_i\} = \frac{f_i(s)P_\pi(s)}{\sum_{s'} f_i(s')P_\pi(s')} \tag{31}$$

Then we modify Equation 26 by adding a summation over the detector regions which are active after the action is taken, instead of simply assuming that some region $S_{s'}$ is the only active region:

$$v_0^\pi(s,a) = \sum_{s'} \mathcal{P}_{ss'}^a \left(\mathcal{R}_{ss'}^a + \gamma \sum_j f_j(s') \sum_{a'} \tau(S_j, a') v_0^\pi(S_j, a') \right) \tag{32}$$

This equation calculates the reward for a policy τ, which may be different from π, the sampling policy. We usually want to know the values for an optimal policy; in that case, the equation becomes

$$v_0^\pi(s,a) = \sum_{s'} \mathcal{P}_{ss'}^a \left(\mathcal{R}_{ss'}^a + \gamma \sum_j f_j(s') \max_{a'} v_0^\pi(S_j, a') \right) \tag{33}$$

This is the result given by Singh, Jaakkola and Jordan, but in different notation. Note that these results are consistent with the earlier formulas we derived for the case of a partition. In particular, if our representation is a partition, Equation 31 then becomes

$$\Pr\{s|S_i\} = \frac{P_\pi(s)}{\sum_{s' \in S_i} P_\pi(s')}$$

because $f_i(s) = 1$ for $s \in S_i$, but is zero for any other states. Equation 33 is equivalent to Equation 27 in the case of a partition, because $f_j(s') > 0$ only for a single region, the one we called $S_{s'}$ in the earlier equations.

4.4.2 Method 2: Exploit assumptions of error minimization

In a discrete-state representation, each action is represented by a separate action value for each state. But in a representation which generalizes over states, the number of parameters available for storing those action values may be much smaller than the

number of states. Therefore, each action value will represent the expected contributions of multiple states, making it a compromise between the different values which would hold for the individual states. Thus any set of generalized action values will be in error for some of the states. This means that there is no one "true" set of action values under state generalization, but only better and worse compromises. We want to define the values so that the combined effect of the errors is minimal. But what does this mean?

Possible error functions

In order to minimize the errors, we have to decide which errors matter most, and how to weight them in order to come up with a function for the combined error, which we can then minimize. There are many ways of doing this. For example, we could select a representative state for each region; we might choose to only count the errors for these prototype states, or we might weight the error for each state by the distance of that state from the prototype for its region. Or we could choose to weight the errors by some other assessment of the relevance of particular states in our task, such as their frequency.

Since we have a separate set of parameters for each action a_k, we may proceed by minimizing the error separately for each action. Let $\mathrm{Err}(k)$ represent the error for action a_k. The normal way of defining such an error function is to sum the squared errors in action value over the set of states. This requires us to define a target for the desired values of the parameters. We will take $v_1^*(s, a_k)$ as the target for w_{ik}. By making the whole-path assumption, we get the actual value the agent would experience, not the apparent value, which could be distorted by the representation's generalization of future states. Thus the whole-path assumption allows us to express the theoretical

action values for any representation in terms of the values the agent would experience under a discrete representation, enabling us to see how the representation has affected the action values. Note that we could also use $v_0^\pi(s, a_k)$ if we wanted to assume the one-step definition of observed values, as in Q-learning. Our target value represents the expectation of the agent's return for taking action a_k from state s and following an optimal policy thereafter.

If we regard the parameter w_{ik} as the expected return for action a_k from the states which are detected by the associated detector, f_i, then we might sum the errors over states as follows:

$$\text{Err}(k) = \sum_i \sum_s (v_1^*(s, a_k) - w_{ik})^2 f_i(s) \tag{34}$$

Notice that Equation 34 sums the errors "democratically," giving equal weight to each state. If we decide that our error equation should give more emphasis to states which occur more frequently, we could include the probability of the state under our sampling policy, π :

$$\text{Err}(k) = \sum_i \sum_s P_\pi(s)(v_1^*(s, a_k) - w_{ik})^2 f_i(s) \tag{35}$$

Equations 34 and 35 only compare the value of w_{ik} against the expected returns for the set $\mathcal{S}_{f_i}$, since $f_i(s) = 0$ everywhere else. In essence, these equations sum the "regional errors" of the action values. If we care most about the total system error, we might define the error as the difference between the true return for (s, a_k) and our system estimate, $Q(s, a_k) = \sum_i w_{ik} f_i(s)$. This can take the form of the familiar mean-squared error equation:

$$\text{Err}(k) = \sum_s P_\pi(s)(v_1^*(s, a_k) - Q(s, a_k))^2 \tag{36}$$

Thus there are many ways of defining the error in the action values. We choose an

error function according to our prior assumptions about which errors matter most. If our approach is to update the system parameters according to a gradient-descent in the error, our choice of error function will also determine which update rule we use, and therefore, how the action values are learned. The point to see here is that the assumptions which lead us to a particular error equation also indicate the way the action values must be set in order to make the best compromises in the action values. Thus the choice of learning algorithm depends on the same assumptions.

Finding weights which minimize the error

Therefore, we may define generalized action values by carefully choosing an error function, and then finding the weight parameters which minimize the error. Let $\hat{\mathbf{w}}_k$ denote the vector of weight parameters which minimize the error for the action a_k, and let $\hat{w}_{ik}$ represent the corresponding weight on the detector f_i. Then the *apparent* action values under a particular representation are given by $Q(s, a_k) = \sum_i \hat{w}_{ik} f_i(s)$. Comparing these values with the discrete-state values $v_1^*(s, a_k)$ will indicate how the representation has changed the action values.

Suppose our error is defined by the "regional errors" function of Equation 35. Then the derivative of the error with respect to the weight w_{ik} is

$$\frac{\partial \mathrm{Err}(k)}{\partial w_{ik}} = \beta \sum_s P_\pi(s) \left(v_1^*(s, a_k) - w_{ik}\right) f_i(s) \tag{37}$$

By setting the derivatives to zero, we find that the weights which represent the best compromise in the action values are given by

$$\sum_s P_\pi(s) v_1^*(s, a_k) f_i(s) = \sum_{s'} P_\pi(s') w_{ik} f_i(s')$$

and, taking w_{ik} outside the sum,

$$\hat{w}_{ik} = \frac{\sum_s P_\pi(s) f_i(s) v_1^*(s, a_k)}{\sum_{s'} P_\pi(s') f_i(s')} \tag{38}$$

Thus $\hat{w}_{ik}$ is a weighted average of the expected rewards, $v_1^*(s, a_k)$, for states $s \in \mathcal{S}_{f_i}$. It is interesting to note that in the case of a partition, the weight on $v_1^*(s, a_k)$ becomes

$$\frac{P_\pi(s)}{\sum_{s' \in S_i} P_\pi(s')}$$

for $s \in \mathcal{S}_{f_i}$, and 0 otherwise. Therefore, in the case of a partition, our "regional errors" function yields the following result:

$$\hat{w}_{ik} = \sum_s \Pr\{s | \mathcal{S}_{f_i}\} v_1^*(s, a_k) \tag{39}$$

This agrees with the result given by our earlier analysis in Equation 23.

Now suppose our error is defined by the "system error" approach of Equation 36. To simplify the notation, let us express the derivative by vector notation, with bold-faced letters representing column vectors, and the transpose of a vector $\mathbf{w}$ written $\mathbf{w}^t$. Then

$$Q(s, a_k) = \sum_i w_{ik} f_i(s) = \mathbf{w}_k^t \mathbf{f}(s)$$

and the derivative of the error with respect to $\mathbf{w}_k$ then becomes

$$\frac{\partial \mathrm{Err}(k)}{\partial \mathbf{w}_k} = \beta \sum_s P_\pi(s) \left(v_1^*(s, a_k) - \mathbf{w}_k^t \mathbf{f}(s) \right) \mathbf{f}(s) \tag{40}$$

Setting this derivative to zero, we have

$$\sum_s P_\pi(s) v_1^*(s, a_k) \mathbf{f}(s) = \sum_{s'} P_\pi(s') \left(\mathbf{w}_k^t \mathbf{f}(s') \right) \mathbf{f}(s')$$

We know from linear algebra that whenever we have two vectors $\mathbf{x}$ and $\mathbf{y}$,

$$\left(\mathbf{x}^t \mathbf{y} \right) \mathbf{y} = \left(\mathbf{y} \mathbf{y}^t \mathbf{x} \right) = \left(\mathbf{y} \mathbf{y}^t \right) \mathbf{x}$$

Therefore,

$$\begin{aligned}\sum_s P_\pi(s)\upsilon_1^*(s,a_k)\mathbf{f}(s) &= \sum_{s'} P_\pi(s')\left(\mathbf{f}(s')\mathbf{f}^t(s')\right)\mathbf{w}_k \\ &= \left(\sum_{s'} P_\pi(s')\mathbf{f}(s')\mathbf{f}^t(s')\right)\mathbf{w}_k,\end{aligned}$$

since $\mathbf{w}_k$ does not depend on s'. Thus

$$\hat{\mathbf{w}}_k = \left(\sum_{s'} P_\pi(s')\mathbf{f}(s')\mathbf{f}^t(s')\right)^{-1} \sum_s P_\pi(s)\upsilon_1^*(s,a_k)\mathbf{f}(s) \tag{41}$$

We should verify that this definition of $\hat{\mathbf{w}}_k$ is consistent with our previous results for the special case of a partition, as we have already demonstrated for the "regional errors" function. In the case of a partition, the matrix $\left(\sum_{s'} P_\pi(s')\mathbf{f}(s')\mathbf{f}^t(s')\right)^{-1}$ of Equation 41 is a diagonal matrix which we can write as

$$M^{-1} = (m_{ij})^{-1} = \begin{pmatrix} m_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & m_{nn} \end{pmatrix}^{-1} = \begin{pmatrix} 1/m_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1/m_{nn} \end{pmatrix}$$

where $m_{ij} = \sum_s P_\pi(s) f_i(s) f_j(s)$. The matrix is diagonal because a partition never has more than one active region at any time, so that $f_i(s)f_j(s) = 0$ unless $i = j$. Substituting this result into Equation 41, we have

$$\hat{\mathbf{w}}_k = (\ldots, \frac{\sum_s P_\pi(s)\upsilon_1^*(s,a_k) f_i(s)}{m_{ii}}, \ldots)$$

resulting in

$$\hat{w}_{ik} = \frac{\sum_s P_\pi(s) f_i(s) \upsilon_1^*(s,a_k)}{\sum_{s'} P_\pi(s') f_i^2(s')}$$

Since $f_i(s)$ only takes values 0 or 1 in a partition, this equation is equivalent to Equation 38 for the "regional errors" function. As we have seen, this equation yields the same results as Equations 39 and 23 in the case of a partition. Therefore, in the case

of the partitioned representation of Figure 21b, we would again arrive at the values summarized in Tables 5 and 6.

The "regional error" and "system error" functions (Equations 35 and 36, respectively) are just two examples out of many possible functions we could choose. This choice, and the choice between the error minimization and soft state aggregation approaches, determine the meaning of the generalized action values, $\{w_{ik}\}$.

4.4.3 Example: Generalized action values for a coarse coded representation

For either Method 1 or Method 2, we can calculate the generalized action values $\hat{w}_{ik} = \upsilon^{\pi}(\mathcal{S}_{f_i}, a_k))$ and then use them to calculate the action-value function, $Q(s, a_k) = \hat{\mathbf{w}}_k \mathbf{f}$. We will do this now for a coarse-coded representation of the gridworld. The resulting action-value function will allow us to see how state generalization has modified the apparent action values in the task.

Suppose that our representation is made up of two Gaussian-kernel feature detectors, as in Figure 21c. We will define the kernel functions and normalized detector functions as follows:

$$g_i(s) = e^{-(s-c_i)^2/0.1}$$

$$f_i(s) = \frac{g_i(s)}{g_1(s) + g_2(s)}$$

where $c_1 = (2.0, 3.0)$ and $c_2 = (3.0, 2.0)$. Detector f_1 claims the entire top-left corner of the grid, and f_2 dominates the other half, the bottom-right corner.

We will assume that the sampling policy is random, so that we are equally likely to choose action up or right from any state. The policy chosen in the earlier examples (moving up the left edge of the grid and then moving right along the top edge) turns out

to be invalid for this representation. The reason is that neither detector can distinguish the top row from the left edge, since the detectors' centers lie on the diagonal which separates the bottom-left from the top-right halves of the grid. The random policy has another advantage: by allowing us to sample all states, it gives us a comprehensive picture of the action values in this task.

Method 1: Soft state aggregation

With state generalization, the action values depend on the sampling policy. Table 9 gives the state probabilities for a random sampling policy. Given the detector func-

0.023	0.045	0.057	0.000
0.045	0.068	0.068	0.057
0.091	0.091	0.068	0.045
0.182	0.091	0.045	0.023

Table 9: Probability of state occurrence under a random policy

tions and the state probabilities, we first calculate the conditional probabilities $\Pr\{s|S_i\}$ according to Equation 31. Then Equations 30 and 33 allow us to write a system of equations expressing the relationships between the action values for the regions ($\upsilon_0^\pi(S_i, a)$) and for the individual states ($\upsilon_0^\pi(s, a)$). Solving this system for the generalized action values ($\hat{w}_{ij} = \upsilon_0^\pi(S_i, a_j)$ here) gives the values which correspond to the state clusters; plugging these values into our system model $Q(s, a_k) = \hat{\mathbf{w}}_k \mathbf{f}$ then allows us to see how the action values have changed from the true values (which are seen under a discrete representation).

The generalized action values are given by Table 10; Figure 23 shows the resulting function $Q(s, \mathsf{right})$ and Figure 24 shows the decision surface $Q(s, \mathsf{right}) - Q(s, \mathsf{up})$. The function $Q(s, \mathsf{up})$ (not shown) is the mirror image of $Q(s, \mathsf{right})$, reflected in the

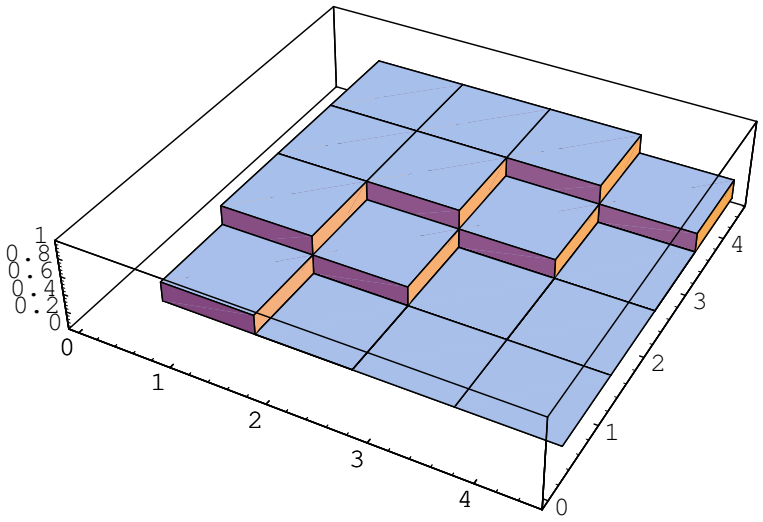


Figure 23: $Q(s, \mathsf{right})$ under soft-state aggregation

diagonal line $x = y$, as expected.

Cluster	$\hat{w}(S_i, \mathsf{up})$	$\hat{w}(S_i, \mathsf{right})$
S_1	0.129213	0.561798
S_2	0.561798	0.129213

Table 10: Generalized action values under soft-state aggregation

Method 2: Minimizing "regional errors"

Instead of solving a system of equations, we calculate the generalized action values directly in terms of the true values for individual states: the $v_1^*(s, a_k)$, which are all powers of the discount parameter, $\gamma = 0.9$, and are summarized below in Table 11 and Table 12. Table 13 summarizes the generalized action values given by this method. Figure 25 shows the function $Q(s, \mathsf{right})$ for this method, and Figure 26 shows the

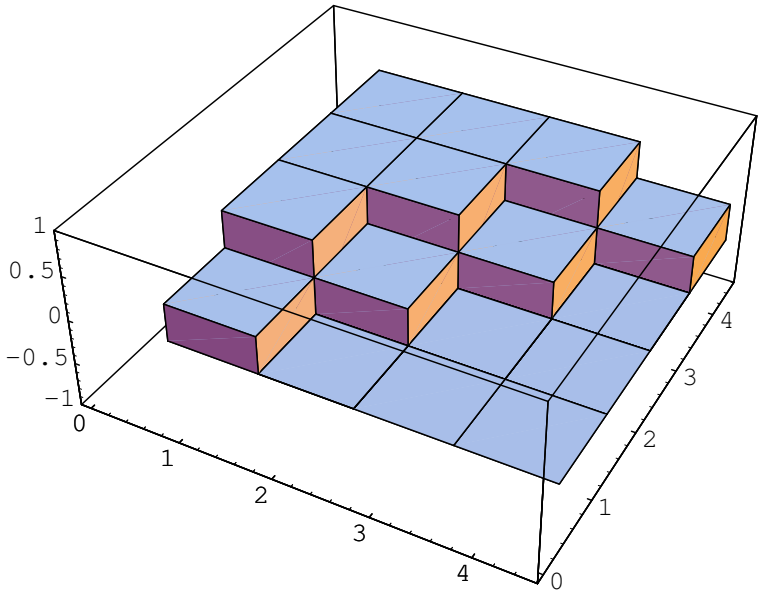


Figure 24: $Q(s, \mathsf{right}) - Q(s, \mathsf{up})$ under soft-state aggregation

0.81	0.9	1.0	0.0
0.729	0.81	0.9	−1.0
0.6561	0.729	0.81	−1.0
0.59049	0.6561	0.729	−1.0

Table 11: Whole-path action values, $v_1^*(s, \mathsf{right})$

−1.0	−1.0	−1.0	0.0
0.729	0.81	0.9	1.0
0.6561	0.729	0.81	0.9
0.59049	0.6561	0.729	0.81

Table 12: Whole-path action values, $v_1^*(s, \mathsf{up})$

decision surface $Q(s, \mathsf{right}) - Q(s, \mathsf{up})$. The function $Q(s, \mathsf{up})$ (not shown) is again the mirror image of $Q(s, \mathsf{right})$.

4.5 The Convex Generalization Property

In both examples, the generalized action values for a detector fell within the range of the action values for the individual states. We might wonder whether this will always be true. If we interpret the detectors as indicators of "generalized states," we might hope that their generalized action values can be expressed as weighted sums of the state-action values. The following criterion articulates this idea more precisely.

Definition 4.1 (Convex Generalization Property (CGP)) *Detector f_i is associated with generalized action values $\hat{w}_{ij}$. The Convex Generalization Property holds*

Cluster	$\hat{w}(S_i, \mathsf{up})$	$\hat{w}(S_i, \mathsf{right})$
S_1	0.281016	0.763289
S_2	0.763289	0.281016

Table 13: Generalized action values under minimization of "regional errors"

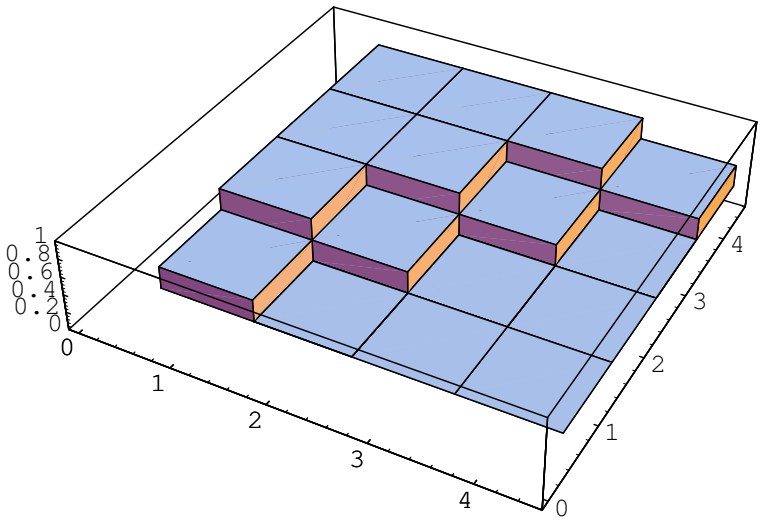


Figure 25: $Q(s, \mathsf{right})$ under minimization of "regional errors"

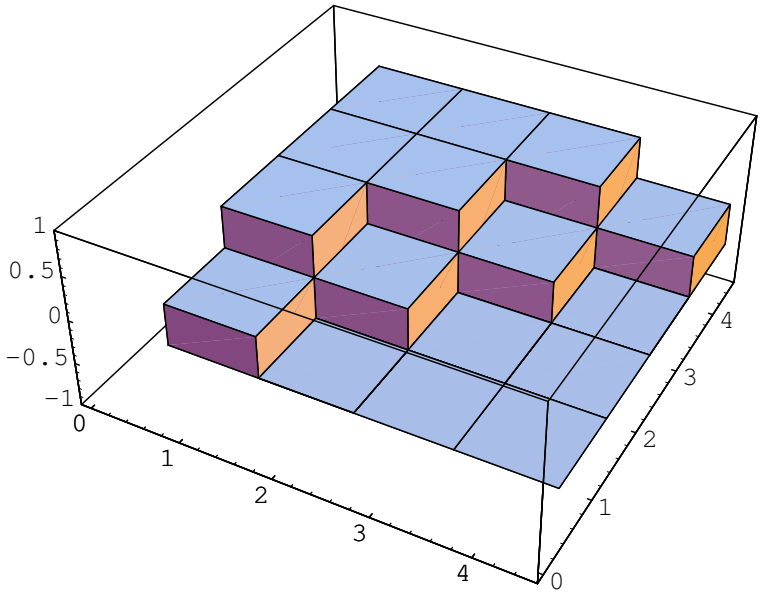


Figure 26: $Q(s, \mathsf{right}) - Q(s, \mathsf{up})$ under minimization of "regional errors"

if

$$\hat{w}_{ij} = \sum_{s:f_i(s)>0} \alpha(s) v(s, a_j) \tag{42}$$

where $v(s, a_j)$ *is the true value of action* a_j *from state* s *under a discrete representation, and the coefficients,* $\alpha(s)$*, obey the following conditions:*

$$\alpha(s) \geq 0, \text{ for } s : f_i(s) > 0$$

$$\sum_{s:f_i(s)>0} \alpha(s) = 1$$

Furthermore, the coefficients, $\alpha(s)$*, do not depend on the action,* a_j*, but are the same for all* j*.*

CGP holds for both soft state aggregation and "regional errors" minimization, but not for "global errors" minimization. For soft state aggregation, Equation 30 expresses the generalized action value $\hat{w}_{ij} = v_0^\pi(S_i, a_j)$ as a weighted sum of state-action values $v_0^\pi(s, a_j)$. In this case, the coefficient $\alpha(s) = \Pr\{s|S_i\}$. Since probabilities are always non-negative, we know that $\alpha(s) \geq 0$. Furthermore,

$$\sum_{s:f_i(s)>0} \alpha(s) = \sum_s \alpha(s)$$

because $\Pr\{s|S_i\} = 0$ when $f_i(s) = 0$. And by Equation 31 we know that

$$\sum_s \alpha(s) = \frac{\sum_s f_i(s) P_\pi(s)}{\sum_{s'} f_i(s') P_\pi(s')} = 1$$

Finally, note that the coefficient $\alpha(s)$ has no dependence on the action, since $\Pr\{s|S_i\}$ is the same, no matter which action a_j is considered in Equation 30. Thus the contribution of $v^\pi(s, a)$ depends purely on the state clustering and the sampling policy.

In the case of regional errors minimization, Equation 38 expresses $\hat{w}_{ik}$ as a linear combination of the terms $v_1^*(s, a_k)$, with coefficients

$$\alpha(s) = \frac{P_\pi(s) f_i(s)}{\sum_{s'} P_\pi(s') f_i(s')}$$

Since the probabilities $P_\pi(s)$ must be non-negative, and we know $f_i(s) \geq 0$ by definition, then $\alpha(s) \geq 0$. Also note that

$$\sum_{s:f_i(s)>0} \alpha(s) = \sum_{s:f_i(s)>0} \frac{P_\pi(s)f_i(s)}{\sum_{s'} P_\pi(s')f_i(s')} = \frac{\sum_s P_\pi(s)f_i(s)}{\sum_{s'} P_\pi(s')f_i(s')} = 1$$

And the coefficients $\alpha(s)$ again have no dependence on the actions or action values. Thus CGP holds for this approach.

But what of the global error minimization approach? Equation 41 expresses the vector of generalized action values, $\hat{\mathbf{w}}_k$, as a linear combination of the $v_1^*(s, a_k)$ but the coefficients $\alpha(s)$ may sometimes be negative because they depend on the inverse matrix M^{-1}. If CGP holds, then $\hat{w}_{ij}$ should be a convex combination of the action values for individual states, and $\hat{w}_{ij}$ should be bounded by those values. (The appendix to Chapter 5 discusses properties of convex combinations in more detail). But the following counter-example shows that $\hat{w}_{ij}$ can be outside the range of the $v_1^*(s, a_k)$, if we take the global error minimization approach.

In this example, the representation is made up of three Gaussian-kernel detectors: one broad detector, centered in the grid, and two narrow detectors placed in the middle of the top row and in the middle of the right column. Figure 27 shows a superposition of the three detectors. In this demonstration we also change the reward function so that falling off the top or right edge of the grid is now given a small, positive reward of 0.1. Entering state (4, 4) results in a reward of 1.0, as before. We calculate the action values by the global error minimization method; the decision surface $Q(s, \mathsf{right}) - Q(s, \mathsf{up})$ is shown in Figure 28. Although it is hard to see, the value for $((1, 4), \mathsf{right}) - ((1, 4), \mathsf{up}) = 0.000000006$, so that the preferred policy in $(1, 4)$ is to move right. Thus the representation "works" for solving the task; but the generalized action values are outside the range occupied by the action values for the individual states, as shown by

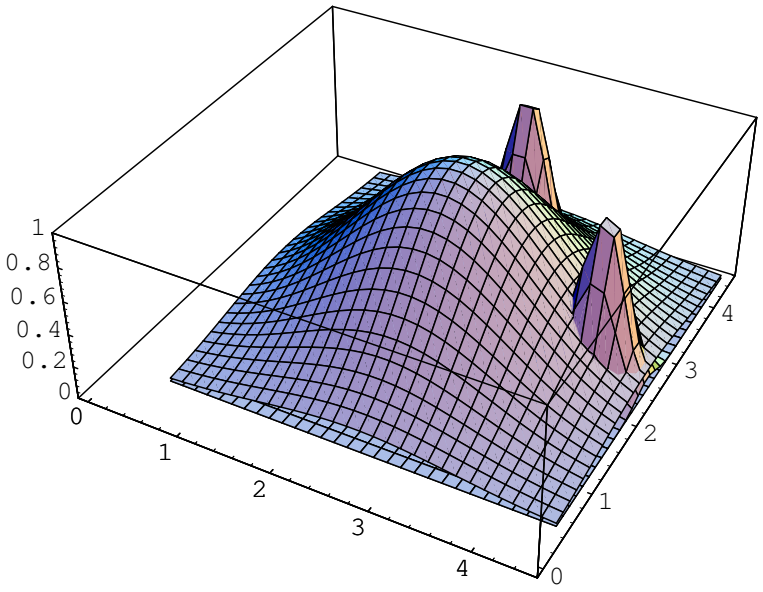


Figure 27: Superposition of the detectors for the counter-example

Table 14. The rewards given in the task are either 0.1 or 1.0—but the generalized action values are sometimes greater than 1.8 or less than −1.9. This shows that CGP does not always hold under global error minimization.

Cluster	$\hat{w}(S_i, \mathsf{up})$	$\hat{w}(S_i, \mathsf{right})$
S_1	−1.97172	1.87002
S_2	1.87002	−1.97172
S_3	0.693563	0.693563

Table 14: Generalized action values under minimization of "global errors"

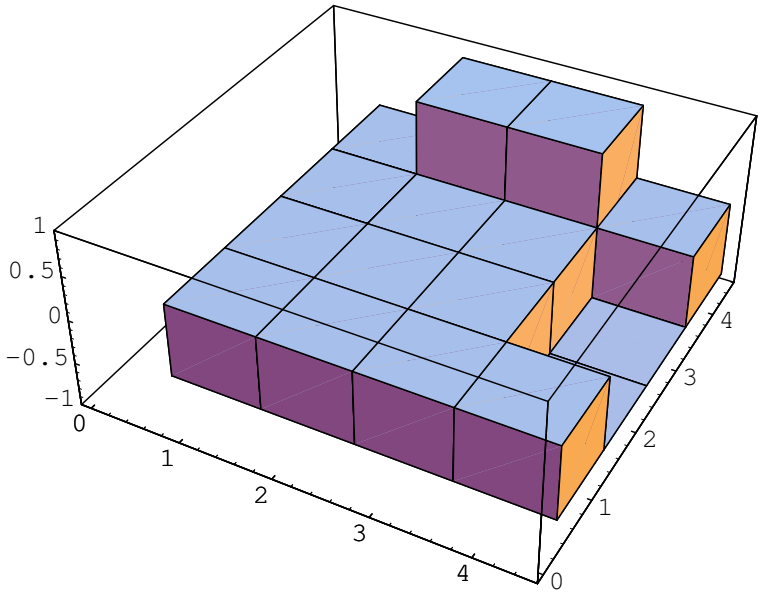


Figure 28: $Q(s, \mathsf{right}) - Q(s, \mathsf{up})$ under minimization of "global errors"

4.6 Effects of Representation on Action Values

How we choose to define generalized action values depends on our purpose in doing so. Soft state aggregation describes the behavior of the Q-learning algorithm, and was used by Singh, Jaakkola and Jordan (1995) to describe the convergence of Q-learning under state generalization. This approach assumes the one-step look-ahead definition of expected reward, and makes the assumption that each detector computes a probability that the current state belongs to the corresponding cluster.

The error minimization approach allows more flexibility in defining the meaning of generalized action values, because the error function allows us to determine which errors are most important, how they should be weighted, and far we look ahead for the action values of individual states. Under this approach we are free to interpret the activation of multiple feature detectors as features which are certain but fuzzy—and therefore, true to different degrees.

The Convex Generalization Property describes a common intuition that the action values of a generalized state ought to derive from the values of the individual states that it covers. Both the soft state aggregation approach and the minimization of "regional" errors satisfy CGP. This property will be a useful tool in the next chapter.

The generalized action values enable us to see how state generalization modifies the apparent value of the agent's actions in the task. By defining the theoretical values for $\hat{w}_{ij}$, we can discuss the effect of representation apart from consideration of the actual learning algorithm in use, or the conditions required for convergence of the weights. Previous sections calculated the generalized action values according to several different schemes. To evaluate these results, we should compare them with the true action values under a discrete representation. Tables 11 and 12 gave the true values for the discrete

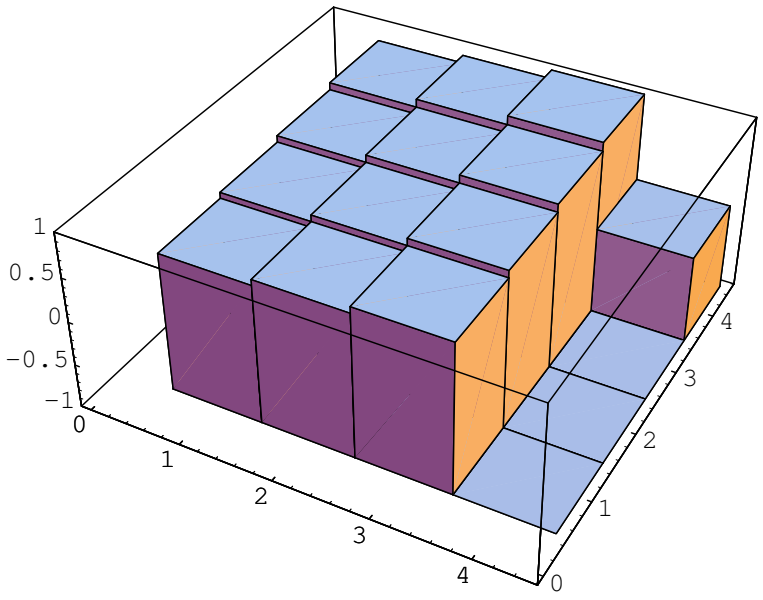


Figure 29: $Q(s, \mathsf{right})$ for the discrete representation

representation; Figure 29 plots these values for the action right, and Figure 30 shows the resulting decision surface $Q(s, \mathsf{right}) - Q(s, \mathsf{up})$.

Comparing these plots with the earlier plots, we see that state generalization has distorted the decision surface, although not enough to prevent the agent from succeeding in the task. The generalized states represent the task with only two sets of action values, instead of 16 for the discrete representation. Such simplification can make a task much easier to learn, as long as the representation preserves the information relevant to the task.

Another observation we might make is that the decision surfaces produced by different representations vary in effectiveness. For example, the decision surface for the discrete representation (Figure 30) has a larger range between its extreme values than

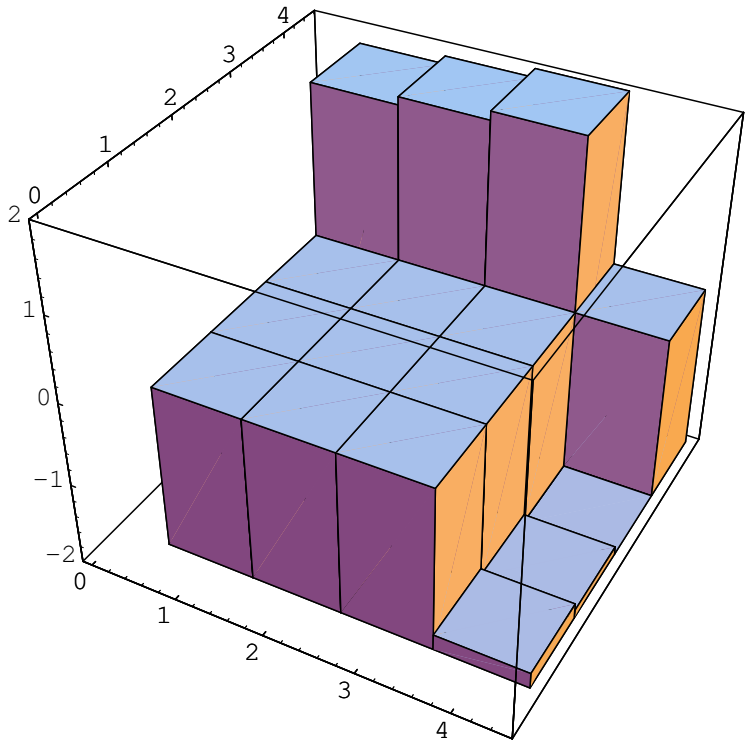


Figure 30: $Q(s, \mathsf{right}) - Q(s, \mathsf{up})$ for the discrete representation

the decision surfaces for the other representations, making the relative advantage of one action over another easier to learn. Whether this apparent increase in feature importance outweighs the added complexity of the representation depends on the task. These two aspects of cognitive economy are in constant tension, so that feature importance is not the only consideration; however, there are some state distinctions which *must* be made by the representation if the task is to remain learnable. The next chapter shows how separating incompatible states preserves the learnability of the task.

4.7 Summary

This chapter extended the idea of action value to the definition of action values for generalized states. The chapter presented definitions of generalized action values based on two different sets of assumptions. These definitions show how state generalization results in compromises in the action values—in essence, how the way we classify the world changes the nature of the tasks we face.

Chapter 5

State Compatibility and Representational Adequacy

> The aim of a model is, of course, precisely not to reproduce reality in all its complexity. It is rather to capture in a vivid, often formal, way what is essential to understanding some aspect of its structure or behavior. The word "essential" as used in the above sentence is enormously significant, not to say problematical. It implies, first of all, purpose. [...] We select, for inclusion in our model, those features of reality that we consider to be essential to our purpose.
>
> —*Joseph Weizenbaum*

5.1 Introduction

This chapter is devoted to a theorem which proves that the state compatibility criteria developed in Chapter 3 lead to representations that allow the agent to learn to make sound decisions in its task. In other words, if each pair of incompatible states is separated in the representation, the representation will be ϵ-adequate. Although the on-line aspects of the task make determination of necessary and sufficient conditions for learnability especially difficult, we can show that the state compatibility criteria are

sufficient conditions for the adequacy of the representation, which is our standard for the learnability of the task.

Although the applicability of the proof is limited to systems which represent incompatible states orthogonally (as partitions do), this result adds support to the claim that these criteria reflect some essential characteristics of important features, and provide a solid foundation for future implementations of on-line learning systems.

5.1.1 Summary of notation

To prove the theorem, we must take care to distinguish properties of our current state, s, from those of the generalized state which contains s. The functions Q, V, and pref_ϵ describe the values and policy for the generalized state, while their one-step look-ahead counterparts ($Q1, V1$, and pref1_ϵ) describe the action values and policy of the individual state, s. These functions were developed in Chapter 3, but for convenience, this section restates their definitions, as well as the criteria for ϵ-adequacy and state compatibility.

We also need to be able to refer to the action which appears best from state s, and the action which appears best for the generalized state. After summarizing previous definitions, the section defines these two concepts of the best action.

In this chapter, I will write $\hat{w}_{ij}$ to refer to the true, steady-state version of the generalized action value w_{ij}, after the analysis in Chapter 4.

Definition 5.1 ($Q(s,a)$) *We define the action value function Q in terms of the generalized action values and feature detectors:*

$$Q(s, a_k) = \sum_i \hat{w}_{ik} f_i(s)$$

Definition 5.2 ($V(s)$) *We define the state value function V in terms of the action values for the generalized state:*

$$V(s) = \max_k Q(s, a_k)$$

Definition 5.3 ($\mathbf{pref}_\epsilon(s)$) *Let s be a state, and let $a_i \in \mathcal{A}(s)$ be an action available to the agent from state s. Then we define*

$$pref_\epsilon(s) = \{a \in \mathcal{A}(s) : Q(s, a) \geq \max_i Q(s, a_i) - \epsilon\}$$

Thus $\text{pref}_\epsilon(s)$ is the set of "best actions" for the agent at s, according to the generalized state.

Definition 5.4 ($Q1(s, a)$) *We define the function $Q1$ for the value of an action, based on the expected discounted future reward seen one step in the future. Let s be the state seen by the agent at time t, and s' a possible resulting state at time $t + 1$. We define*

$$Q1(s, a) = \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$$

where

$$V(s') = \max_j Q(s', a_j) = \max_j \sum_i \hat{w}_{ij} f_i(s')$$

Definition 5.5 ($V1(s)$) *Let s be a state, and $a_i \in \mathcal{A}(s)$ an action available to the agent from state s. Then we define*

$$V1(s) = \max_i Q1(s, a_i)$$

Thus $V1(s)$ represents the expected value of state s, based on a one-step look-ahead from s.

Definition 5.6 (pref1$_\epsilon(s)$) *Let s be a state, and let $a_i \in \mathcal{A}(s)$ be an action available to the agent from state s. Then we define*

$$pref1_\epsilon(s) = \{a \in \mathcal{A}(s) : Q1(s,a) \geq \max_i Q1(s,a_i) - \epsilon\}$$

Thus pref1$_\epsilon(s)$ is the set of "best actions" for the agent at s, according to a one-step look-ahead.

Definition 5.7 (ϵ-adequacy) *Let δ be given, and assume that when the agent's current state is s_t it always selects an action from $pref_\delta(s_t)$. We will say that a representation of the state-space is an ϵ-adequate representation for $\delta \leq \epsilon$, if for every state $s_t \in \mathcal{S}$ reachable by the agent, the following two properties hold:*

$$|V1(s_t) - V(s_t)| \leq \epsilon$$

$$pref_\delta(s_t) \subseteq pref1_\epsilon(s_t)$$

Definition 5.8 (State compatibility) *Let s_1 and s_2 be states in $\mathcal{S}$. Let δ be given, and assume that the agent always selects an action from $pref_\delta(s_t)$. Also assume that our goal is to produce an ϵ-adequate representation, where $\epsilon \geq \delta$.*

We will say that s_1 and s_2 are compatible *in case the following three conditions hold:*

$$pref1_\epsilon(s_1) = pref1_\epsilon(s_2) \tag{43}$$

$$|V1(s_1) - V1(s_2)| \leq \delta \tag{44}$$

and

$$pref1_\delta(s_1) = pref1_\delta(s_2) \quad \text{if } \delta \leq \epsilon/2 \tag{45}$$

$$pref1_0(s_1) = pref1_0(s_2) \quad \text{otherwise} \tag{46}$$

Definition 5.9 (Best action at individual state) *Let $a^*(s)$ denote the action that is best according to a one-step look-ahead for the state s :*

$$a^*(s) = \arg\max_a Q1(s, a)$$

Definition 5.10 (Best action for generalized state) *Let $\hat{a}$ denote the apparent best action at s, according to the generalized action values:*

$$\hat{a} = \arg\max_a Q(s, a)$$

Note that since $\hat{a}$ depends only on the values for the generalized state, it is the same for every member of that generalized state. If we choose any two states s_1 and s_2 in the same generalized state, we may find that $a^*(s_1) \neq a^*(s_2)$, but the generalized action values still produce $\hat{a}$ as the apparent best action for both. When $\hat{a} \neq a^*(s)$, this means that state generalization has resulted in compromises in the action values, so that $\hat{a}$, the action which appears best for the generalized state, is not the best action for the state s, according to a one-step look-ahead.

Although $\hat{a}$ is technically a function of s, I will refer to it simply as $\hat{a}$. Since the following analysis will always consider a single generalized state at a time, we will know which $\hat{a}$ is meant, and this notation will reinforce the difference between $a^*(s)$, which is specific to s, and $\hat{a}$, which is common to all states grouped with s.

5.1.2 Examples of what can go wrong

The examples which follow illustrate what can go wrong when the compatibility rules are violated. For simplicity, assume that each example presents a set of states from the same region in a partition representation; assume also that the states have equal

probability of occurrence, so that the action values $Q(s, a_j)$ may be found by simply averaging the $Q1(s_i, a_j)$ values for the individual states.

In general, $a^*(s_i)$ could be a different action for each s_i. Thus $\hat{a}$, which appears best for the generalized state, may not be best for all the states in this grouping—or even for any the states. If the values of the actions vary widely from state to state, $\hat{a}$ may be a poor enough choice for some states that the representation is not ϵ-adequate. That is why we need the assumption regarding the consistency of the top action values (Equations 45 and 46). Table 15 shows a system of three states which satisfies the first two compatibility rules, but not the third. In this example, $\delta \leq \epsilon/2$, but there is no

a_i	$Q1(s_1, a_i)$	$Q1(s_2, a_i)$	$Q1(s_3, a_i)$	$Q(s, a_i)$
a_1	1.00	0.80	0.80	$0.8\overline{6}$
a_2	0.80	1.00	0.80	$0.8\overline{6}$
a_3	0.80	0.80	1.00	$0.8\overline{6}$
a_4	0.79	0.79	0.79	0.79
$V1(s_j)$	1.00	1.00	1.00	$V(s) = 0.8\overline{6}$
$\text{pref1}_\epsilon(s_j)$	$\{a_1, a_2, a_3\}$	$\{a_1, a_2, a_3\}$	$\{a_1, a_2, a_3\}$	

Table 15: Inconsistency of action rankings may prevent ϵ-adequacy ($\delta = 0.08, \epsilon = 0.2$). Here $\text{pref}_\delta(s) = \{a_1, a_2, a_3, a_4\}$

action whose value is always within δ of the best action value for each state. In other words, for any action a_i there is a state s such that $Q1(s, a^*(s)) - Q1(s, a_i) > \delta$. Even though the state values are compatible (in fact, they are identical), and the $\text{pref1}_\epsilon(s_j)$ are the same for each state, the action a_4 appears in $\text{pref}_\delta(s)$ but not in $\text{pref1}_\epsilon(s_j)$, violating the conditions for ϵ-adequacy. Thus the agent could select an action which appears optimal, and incur an incremental regret greater than ϵ.

Even if the top actions are consistent according to Equation 45, the representation might not be ϵ-adequate for $\epsilon/2 < \delta \leq \epsilon$. For example, consider the system shown in Table 16. In this example, the first two compatibility rules are satisfied, since s_1 and

a_i	$Q1(s_1, a_i)$	$Q1(s_2, a_i)$	$Q(s, a_i)$
a_1	1.00	0.85	0.925
a_2	0.85	1.00	0.925
a_3	0.79	0.79	0.79
$V1(s_j)$	1.00	1.00	$V(s) = 0.925$
$\text{pref1}_\epsilon(s_j)$	$\{a_1, a_2\}$	$\{a_1, a_2\}$	$\text{pref}_\delta(s) = \{a_1, a_2, a_3\}$

Table 16: Inadequate representation for $\delta > \epsilon/2$ ($\delta = 0.15, \epsilon = 0.2$.)

s_2 share the same state values and pref1_ϵ sets. The less rigorous form of the third rule is also satisfied (Equation 45), since $\text{pref1}_\delta(s_1) = \text{pref1}_\delta(s_2) = \{a_1, a_2\}$. But action a_3 appears in $\text{pref}_\delta(s)$ and not in $\text{pref1}_\epsilon(s_1)$ or $\text{pref1}_\epsilon(s_2)$; hence the agent might be led to select a_3 as its action, and incur an incremental regret greater than ϵ. This example shows that ϵ-adequacy requires a stronger condition than Equation 45 when $\delta > \epsilon/2$. For large δ, Equation 46 makes sure that all the states in the group agree as to the best action.

We might be tempted to allow a larger tolerance for the compatibility of state values in Equation 44, perhaps allowing the $V1$ values of two compatible states to be ϵ apart. But then we will no longer be able to guarantee ϵ-adequacy. This is shown by the system in Table 17, which satisfies the compatibility rules, except that the $V1$ values of different states are ϵ apart, instead of being within the tolerance δ. In this example,

a_i	— $Q1(s_j, a_i)$ —						$Q(s, a_i)$
a_1	1.00	0.70	0.80	0.70	0.80	0.70	$0.78\overline{3}$
a_2	0.90	0.80	0.70	0.80	0.70	0.80	$0.78\overline{3}$
a_3	0.40	0.40	0.40	0.40	0.40	0.40	0.40
$V1(s_j)$	1.00	0.80	0.80	0.80	0.80	0.80	$V(s) = 0.78\overline{3}$

Table 17: State value incompatibilities. ($\delta = 0.1, \epsilon = 0.2$)

the pref1 sets are identical for all the states, whether the tolerance used is δ or ϵ. The

state values are all within $\epsilon = 0.2$. But

$$V1(s_1) - V(s_1) = 1.0 - 0.78\overline{3} > 0.2 = \epsilon$$

Hence the representation is not ϵ-adequate.

The compatibility conditions include two rules involving the compatibility of pref1 sets, but with different tolerances. We have seen (Table 15) that a region of states which meets the first of these rules (Equation 43) but not the second (Equation 45 or 46) may fail to be ϵ-adequate. We might wonder if meeting the second rule could allow us to drop the requirement of meeting the first rule. But the example in Table 18 shows that the representation can still fail to be ϵ-adequate when Equation 45 is met but Equation 43 is not.

a_i	$Q1(s_1, a_i)$	$Q1(s_2, a_i)$	$Q1(s_3, a_i)$	$Q(s, a_i)$
a_1	1.00	0.80	0.80	$0.8\overline{6}$
a_2	0.90	0.90	0.80	$0.8\overline{6}$
a_3	0.90	0.80	0.90	$0.8\overline{6}$
a_4	0.79	0.79	0.79	0.79
$V1(s_j)$	1.00	0.90	0.90	$V(s) = 0.8\overline{6}$
$\text{pref1}_\epsilon(s_j)$	$\{a_1, a_2, a_3\}$	$\{a_1, a_2, a_3, a_4\}$	$\{a_1, a_2, a_3, a_4\}$	

Table 18: Inadequate representation where Equation 43 is not met ($\delta = 0.1, \epsilon = 0.2$). Here $\text{pref}_\delta(s) = \{a_1, a_2, a_3, a_4\} \not\subseteq \{a_1, a_2, a_3\} = \text{pref1}_\epsilon(s_1)$.

In this case, $\text{pref}_\delta(s_1) \not\subseteq \text{pref1}_\epsilon(s_1)$. This violates the ϵ-adequacy criteria; as a result, action a_4 appears to be a sound choice in state s_1, even though it incurs an incremental regret greater than ϵ.

5.2 Sufficient conditions for ϵ-adequacy

5.2.1 Generalization of state separation

The reason we need criteria for state-compatibility is to decide when the representation must separate states, in order to prevent the generalization of action values over dissimilar states. But what does it mean to separate two states? In a partition representation, two states are separated if they lie in different partition regions. In this case, the values of one region are completely separate from those of every other region; thus two states either lie in different regions with distinct sets of action values, or else they share the same action values because they are, in effect, aliases for the same partition region. In contrast, the general case allows a state to be represented by a *pattern* of activation over a set of feature detectors, instead of by the single detector for a unique state-space region which covers that state.

Suppose we have two states, s_1 and s_2. If some detector f_i is active for both states, then the generalized action values $\hat{w}_{ij}$ for f_i contribute to both $Q(s_1, a_j)$ and $Q(s_2, a_j)$. This means that experiences with s_1 and s_2 both affect the generalized action values $\hat{w}_{ij}$; therefore, both states affect the action values of the other. Generalization over incompatible states may sometimes be benign, depending on the number of states sharing values, their relative frequencies of occurrence, and whether their differences tend to balance and cancel each other. For example, if most of the feature detectors which are active for s_1 are inactive for s_2, the interaction between them might not matter. Or if the cross-talk between updates for different states tends to cancel because the errors for individual states appear to be random, that cross-talk might not be a problem. Or if the states which result in cross-talk to the action-value updates for

s_1 occur only infrequently, this interaction might not compromise the accuracy of s_1's action values in any significant way.

So if our analysis allows for interaction and cross-talk between incompatible states, then all these factors will have a bearing on our separation criteria. How completely the states need to be separated in the feature space will depend on the trade-offs between state frequencies, the magnitude of the differences in action values, the number and strength of detectors which overlap, and the distribution of the cross-talk errors for individual states. This is beyond the scope of our current discussion. For the purpose of completing the analysis in this chapter, I will simply define *separated states* to be states which have orthogonal representations in feature space. Therefore, there will be no interaction or dependence between the action values of separated states.

Definition 5.11 (State separation) *Given two states, s_1 and s_2, we will say that s_1 and s_2 are* separated *in case* $\mathbf{f}(s_1)\mathbf{f}(s_2) = 0$.

This definition is a generalization of the partition case, where each state excites a single detector, and separated states lie in different partition regions. To see this, suppose that s_1 and s_2 are states which are separated by a partition representation. Let f_{s_1} be the detector which responds to s_1, and f_{s_2} the detector which responds to s_2. Then

$$f_i(s_1) = \begin{cases} 1 & \text{if } f_i = f_{s_1} \\ 0 & \text{otherwise} \end{cases}$$

and

$$f_i(s_2) = \begin{cases} 1 & \text{if } f_i = f_{s_2} \\ 0 & \text{otherwise} \end{cases}$$

Since $f_{s_1} \neq f_{s_2}$, we have $f_i(s_1)f_i(s_2) = 0, \forall i$ and $\mathbf{f}(s_1)\mathbf{f}(s_2) = 0$.

Notice that for the partition representation, a detector cannot be active for both of a pair of separated states. This fact will also hold for the general case, as shown in the following lemma.

Lemma 5.1 *If s_1 and s_2 are separated,*

$$f_i(s_1)f_i(s_2) = 0, \forall i$$

Proof: The feature detectors always take non-negative values, by the definition of f. Therefore, $f_i(s) \geq 0, \forall i, s$, and $f_i(s_1)f_i(s_2) \geq 0$. By the separation of s_1 and s_2, we know that $\mathbf{f}(s_1)\mathbf{f}(s_2) = \sum_i f_i(s_1)f_i(s_2) = 0$. Therefore, we must have $f_i(s_1)f_i(s_2) = 0, \forall i$.

Corollary 5.1 *If s_1 and s_2 are separated, and $f_i(s_1) > 0$, then*

$$f_i(s_2) = 0 \tag{47}$$

5.2.2 Common assumptions

The theorem to follow will show that if the representation always separates incompatible states, it must be ϵ-adequate. In order to prove this, it makes several assumptions about the task and representation, as well as an assumption about what to do with the portion of the episode beyond the current state. The proof will also draw upon some of the properties of convex combinations, which are summarized at the end of this chapter.

Definition 5.12 (Common Assumptions) *The following assumptions are common to the lemmas and the theorem which follow:*

- *$\mathcal{S}$ is finite*

- *The Convex Generalization Property (see below) holds for our definition of generalized action values $\hat{w}_{ij}$, with a set of coordinates which is independent of the action a_j*

- *The vector of feature detectors is normalized:*

$$\sum_i f_i(s) = 1, \forall s$$

- *The current one-step state-action values are equivalent to the true values:*

$$Q1(s, a_j) = v_0^\pi(s, a_j)$$

Before moving on, let us examine each of these assumptions in more detail.

Finite state-space The arguments in the proofs will involve summations over states. In order to avoid complications resulting from infinite sums, I make the assumption that the state-space either has a finite number of states or that we may replace it with a finite state-space of suitable resolution. If the state-space is infinite we assume that a sufficiently-detailed but finite partitioning of the space exists and provides an adequate representation for the task. (If such a partitioning does not exist, then there are an infinite number of distinctions which must be learned in order to solve the task—but such tasks are ruled out by our assumption that the task is learnable). Therefore, we may assume that there are only a finite number of states $s_i \in \mathcal{S}$, although some of these 'states' may actually cover multiple states from the original representation. Starting with this basic set of states, the purpose of the proof is to show that representations may safely ignore distinctions between compatible states, without compromising the learnability of the task. Thus the representation remains ϵ-adequate.

Convex Generalization Property Assume that the Convex Generalization Property (CGP) holds for the generalized action values. This means that the generalized action values for a set of states (for example, the set of states recognized by a given feature detector) are a special kind of weighted sum of action values for the individual states in the set. As shown in Chapter 4, this property holds for the definitions of generalized action value which interest us most, including "soft state aggregation" (Singh, Jaakkola and Jordan, 1995), as well as my "local minima" definition.

For convenience, we restate the pertinent results for CGP here. Detector f_i is associated with generalized action values $\hat{w}_{ij}$, where

$$\hat{w}_{ij} = \sum_{s:f_i(s)>0} \alpha(s) v^{\pi}(s, a_j) \tag{48}$$

and the coefficients, $\alpha(s)$, obey the following conditions:

$$\alpha(s) \geq 0, \text{ for } s : f_i(s) > 0$$

$$\sum_{s:f_i(s)>0} \alpha(s) = 1$$

Furthermore, the coefficients, $\alpha(s)$, do not depend on the action, a_j, but are the same for all j, as shown in Chapter 4.

Normalization of feature vectors Assume that the sum of the active detectors is unity:

$$\sum_i f_i(s) = 1$$

This is a generalization of discrete Q-learning, (where a single state is active), or learning with partition representations, in which a single detector has value 1 and the others are zero. In "soft state aggregation," Singh, et. al. made this

assumption, because they wished to regard the output of the feature detectors as probabilities that the current state was an example of each cluster-category; thus the detector outputs would need to sum to unity for these values to actually be probabilities. Besides making the proof easier, in practice, the normalization assumption tends to make the system more well-behaved.

Quality of look-ahead values For the purpose of this proof, assume that the one-step look-ahead values are the true action values:

$$Q1(s, a_j) = \upsilon_0^\pi(s, a_j)$$

where π is a policy which results from selecting actions from $\text{pref}_\delta(s)$ for the remainder of the episode.

In Chapter 4 we saw that we have several candidates for the "true" action values for a particular representation. The $\upsilon_0^\pi(s, a_j)$ represent the expected sum of the reward for taking action a_j from s and the discounted value of the resulting state. This is a traditional one-step look-ahead. In contrast, the $\upsilon_1^\pi(s, a_j)$ represent the reward under the "whole-path" assumption—a Monte Carlo return for the rest of the episode. In Chapter 4, the whole-path returns had the advantage of allowing us to see *exactly* how the representation affected the action values and policy for the task; but in this chapter, the one-step assumption is more appropriate, since we want to validate a set of criteria which may be used in feature extraction algorithms which can only see the next step ahead in the episode. Therefore, in the lemmas and theorem to follow, we will take $\upsilon^\pi(s, a_j)$ to mean $\upsilon_0^\pi(s, a_j)$.

This assumption allows us to ignore the rest of the episode and focus on the incremental regret at the current state. The theorem will show that, according

to this definition of action value, separating incompatible states will result in an ϵ-adequate representation. But the proof will not necessarily extend to the case where the true values are considered to be those of a Monte Carlo return.

In order for the representation to be ϵ-adequate, it must be ϵ-adequate at each point in the episode, from the terminal states all the way back to the initial state. If this is the case, then the policy π describes the behavior of an agent which makes sound—although not necessarily optimal—decisions throughout the remainder of the episode.

Lemmas 5.2 and 5.3 relate the action values for the generalized state to the values for individual states. These results will be the foundation for proving the theorem.

Lemma 5.2 *Assume the Common Assumptions of Definition 5.12. Also assume that the representation separates any pair of incompatible states. Let $s \in \mathcal{S}$ and $a_k \in \mathcal{A}$. And let us write $\mathcal{S}_s$ for the set of states which are compatible (by Definition 5.8) with our chosen state, s. Then there exist coefficients $c_{s'}$ which allow us to write $Q(s, a_k)$ as a convex combination of action values for the states s' which are compatible with s, as follows:*

$$Q(s, a_k) = \sum_{s' \in \mathcal{S}_s} c_{s'} \, Q1(s', a_k), \tag{49}$$

$$c_{s'} \geq 0, \sum_{s' \in \mathcal{S}_s} c_{s'} = 1 \tag{50}$$

Proof: Let $s \in \mathcal{S}$. According to our system model,

$$Q(s, a_k) = \sum_i \hat{w}_{ik} f_i(s) \tag{51}$$

By assumption, the f_i are normalized and non-negative. Thus

$$\sum_i f_i(s) = 1, \forall s \in \mathcal{S}$$

$$f_i(s) \geq 0, \forall i, s$$

Hence $Q(s, a_k)$ is a convex combination of generalized action values $\hat{w}_{ik}$. Notice that if $f_j(s) = 0$, the term $\hat{w}_{jk} f_j(s)$ makes no contribution to $Q(s, a_k)$ in Equation 51. Therefore, we may write

$$Q(s, a_k) = \sum_{i: f_i(s) > 0} \hat{w}_{ik} f_i(s) \tag{52}$$

$$\sum_{i: f_i(s) > 0} f_i(s) = 1$$

Hence $Q(s, a_k)$ is a convex combination of the subset of $\{\hat{w}_{ik}\}$ for which $f_i(s) > 0$.

Recall that $\mathcal{S}_{f_i}$ is the *recognition set* for detector f_i and is defined by

$$\mathcal{S}_{f_i} = \{s' \in \mathcal{S} : f_i(s') > 0\}$$

By the Convex Generalization Property (Equation 48), we may write the generalized action value $\hat{w}_{ik}$ as a convex combination of the true action values for states $s' \in \mathcal{S}_{f_i}$:

$$\hat{w}_{ik} = \sum_{s' \in \mathcal{S}_{f_i}} \alpha_{s'} \, v^{\pi}(s', a_k) \tag{53}$$

where the coefficients $\alpha_{s'}$ do not depend on action a_k, as established in Chapter 4.

Note that $\mathcal{S}_{f_i}$ is a pure set, meaning that its members are all mutually compatible states. If not, we could choose $s_1, s_2 \in \mathcal{S}_{f_i}$ such that s_1 and s_2 are not compatible. Because both states are in $\mathcal{S}_{f_i}$ we have $f_i(s_1) > 0$ and $f_i(s_2) > 0$. But if these states are incompatible, they are separated by the representation, by hypothesis. Since we know $f_i(s_1) > 0$, we must have $f_i(s_2) = 0$, by Corollary 5.1—a contradiction. Thus every pair of states from $\mathcal{S}_{f_i}$ must be compatible.

Now if $f_i(s) > 0$, then $s \in \mathcal{S}_{f_i}$. Since $\mathcal{S}_{f_i}$ is a pure set, $f_i(s) > 0$ thus implies that all the members of $\mathcal{S}_{f_i}$ are compatible with our chosen state, s. Therefore

$$f_i(s) > 0 \Rightarrow \mathcal{S}_{f_i} \subseteq \mathcal{S}_s \tag{54}$$

We have already established (Equation 52) that $Q(s, a_k)$ is a convex combination of terms $\hat{w}_{ik}$ for which $f_i(s) > 0$. Furthermore, Equations 53 and 54 show that $f_i(s) > 0$ implies that $\hat{w}_{ik}$ is a convex combination of terms $v^\pi(s', a_k)$ for states $s' \in \mathcal{S}_s$. Since a convex combination of a convex combination of terms is itself a convex combination of those terms (Lemma 5.5), we have established that $Q(s, a_k)$ is a convex combination of terms $v^\pi(s', a_k)$ for states s' which are compatible with s. By our assumption regarding look-ahead values, we may substitute $Q1(s', a_k)$ for $v^\pi(s', a_k)$. Thus there exist coefficients $c_{s'} \geq 0$ such that

$$\sum_{s' \in \mathcal{S}_s} c_{s'} = 1$$

and

$$Q(s, a_k) = \sum_{s' \in \mathcal{S}_s} c_{s'} \, Q1(s', a_k)$$

Lemma 5.3 *Assume the Common Assumptions of Definition 5.12. Also assume that the representation separates any pair of incompatible states. Let $s \in \mathcal{S}$ and $a_j, a_k \in \mathcal{A}$. Then we may write $Q(s, a_j) - Q(s, a_k)$ in terms of action values of states which are compatible with s, as follows:*

$$Q(s, a_j) - Q(s, a_k) = \sum_{s' \in \mathcal{S}_s} c_{s'}(Q1(s', a_j) - Q1(s', a_k)) \tag{55}$$

Proof: This result is a corollary of Lemma 5.2. Notice that the coefficients $c_{s'}$ in Equation 49 do not depend on a_k, since the coefficients in Equation 51 and Equation 53

have no dependence on the action chosen. Thus we may use the same coefficients to write $Q(s, a_j)$ as a convex combination:

$$Q(s, a_j) = \sum_{s' \in \mathcal{S}_s} c_{s'} \, Q1(s', a_j)$$

Thus

$$\begin{aligned} Q(s, a_j) - Q(s, a_k) &= \sum_{s' \in \mathcal{S}_s} c_{s'} \, Q1(s', a_j) - \sum_{s' \in \mathcal{S}_s} c_{s'} \, Q1(s', a_k) \\ &= \sum_{s' \in \mathcal{S}_s} c_{s'} (Q1(s', a_j) - Q1(s', a_k)) \end{aligned}$$

We now have the tools to prove the following theorem, which asserts that the compatibility criteria given in Definition 5.8 are sufficient conditions for ϵ-adequacy.

Theorem 5.1 (State compatibility guarantees ϵ-adequacy)
Assume that the state-space is finite, the Convex Generalization Property holds for our definition of generalized action values $\hat{w}_{ij}$, the feature vectors $\mathbf{f}(s)$ are normalized so that $\sum_i f_i(s) = 1, \forall i, s$, and that the one-step look-ahead values at the current state are equivalent to the true action values: $Q1(s, a_j) = v_0^{\pi}(s, a_j)$. Suppose that the representation separates every pair of incompatible states. Then the representation is ϵ-adequate.

5.2.3 Proof of the theorem

Case 1: $\delta \leq \epsilon/2$

The first step is to establish that for any state $s \in \mathcal{S}, |V1(s) - V(s)| \leq \epsilon$. Since the preceding two Lemmas link $Q(s, a_j)$ with the $Q1$ values for compatible states ($s' \in \mathcal{S}_s$), we will focus on $\mathcal{S}_s$. Let $s_{\max}$ be the state with largest $Q1$ value in $\mathcal{S}_s$, and let $Q1_{\max}$ be that value. Thus

$$Q1_{\max} = V1(s_{\max}) = \max_{s' \in \mathcal{S}_s} V1(s') = \max_{s' \in \mathcal{S}_s} \max_{a_i} Q1(s', a_i)$$

By compatibility of values (Equation 44), we know that

$$|V1(s_{\max}) - V1(s')| \leq \delta, \forall s' \in \mathcal{S}_s$$

Since $V1(s_{\max}) = Q1_{\max}$ and $V1(s') = Q1(s', a^*(s'))$, this is

$$|Q1_{\max} - Q1(s', a^*(s'))| \leq \delta, \forall s' \in \mathcal{S}_s$$

Thus

$$Q1_{\max} - Q1(s', a^*(s')) \leq \delta, \forall s' \in \mathcal{S}_s \tag{56}$$

By compatibility we also know (Equation 45) that there is an action which is close to the best choice for every state in $\mathcal{S}_s$:

$$\exists a_c : a_c \in \text{pref1}_\delta(s'), \forall s' \in \mathcal{S}_s$$

Thus, by the definition of $\text{pref1}_\delta(s')$,

$$Q1(s', a^*(s')) - Q1(s', a_c) \leq \delta, \forall s' \in \mathcal{S}_s$$

Combined with Equation 56, this yields

$$Q1_{\max} - Q1(s', a_c) \leq \delta + \delta \leq \epsilon, \forall s' \in \mathcal{S}_s$$

or

$$Q1(s', a_c) \geq Q1_{\max} - \epsilon, \forall s' \in \mathcal{S}_s \tag{57}$$

By Lemma 5.2 we may write $Q(s, a_c)$ as a convex combination of terms $Q1(s', a_c)$:

$$Q(s, a_c) = \sum_{s' \in \mathcal{S}_s} c_{s'} Q1(s', a_c)$$

Therefore, by applying Equation 57, we have

$$Q(s, a_c) = \sum_{s' \in \mathcal{S}_s} c_{s'} Q1(s', a_c) \geq \sum_{s' \in \mathcal{S}_s} c_{s'} (Q1_{\max} - \epsilon) = Q1_{\max} - \epsilon$$

By the definition of $\hat{a}$, we have $Q(s, \hat{a}) \geq Q(s, a_c)$. Therefore

$$Q(s, \hat{a}) \geq Q1_{\max} - \epsilon \tag{58}$$

But by Lemma 5.2

$$Q(s, \hat{a}) = \sum_{s' \in \mathcal{S}_s} c_{s'} Q1(s', \hat{a})$$

By the definition of $Q1_{\max}, Q1(s', \hat{a}) \leq Q1_{\max}$. Therefore,

$$Q(s, \hat{a}) = \sum_{s' \in \mathcal{S}_s} c_{s'} Q1(s', \hat{a}) \leq \sum_{s' \in \mathcal{S}_s} c_{s'} Q1_{\max} = Q1_{\max}$$

and we have $Q(s, \hat{a}) \leq Q1_{\max}$. Combining this result with Equation 58 yields

$$V(s) = Q(s, \hat{a}) \in [Q1_{\max} - \epsilon, Q1_{\max}] \tag{59}$$

Now consider $V1(s)$. By compatibility of values (Equation 44)

$$|V1(s_{\max}) - V1(s)| \leq \delta$$

Since $V1(s_{\max}) = Q1_{\max}$, this means that $Q1_{\max} - V1(s) \leq \delta < \epsilon$. Therefore

$$V1(s) \in [Q1_{\max} - \epsilon, Q1_{\max}] \tag{60}$$

Since both $V(s)$ and $V1(s)$ are contained by $[Q1_{\max} - \epsilon, Q1_{\max}]$, (Equations 59 and 60, respectively) we know (by Lemma 5.6) that

$$|V1(s) - V(s)| \leq Q1_{\max} - (Q1_{\max} - \epsilon) = \epsilon$$

Thus we have proved the first half of the ϵ-adequacy criterion for the case when $\delta \leq \epsilon/2$.

Now let us establish the second part of the ϵ-adequacy criterion at s : that $\text{pref}_\delta(s) \subseteq \text{pref1}_\epsilon(s)$. We will suppose that $a_k \in \text{pref}_\delta(s)$, but $a_k \notin \text{pref1}_\epsilon(s)$, and show that this

results in a contradiction. By compatibility (Equation 43) we know that $\text{pref1}_\epsilon(s') = \text{pref1}_\epsilon(s), \forall s' \in \mathcal{S}_s$. Therefore

$$a_k \notin \text{pref1}_\epsilon(s'), \forall s' \in \mathcal{S}_s$$

Hence, by the definition of the preference sets, we have

$$Q1(s', a^*(s')) - Q1(s', a_k) > \epsilon, \forall s' \in \mathcal{S}_s \tag{61}$$

By our compatibility assumptions (Equation 45), we know that there is an action, call it a_c, whose one-step look-ahead value is within δ of the top value for *each* compatible state:

$$a_c \in \text{pref1}_\delta(s'), \forall s' \in \mathcal{S}_s$$

Thus

$$Q1(s', a^*(s')) - Q1(s', a_c) \leq \delta, \forall s' \in \mathcal{S}_s$$

and, rearranging the terms,

$$Q1(s', a_c) + \delta \geq Q1(s', a^*(s')), \forall s' \in \mathcal{S}_s$$

Combining this result with Equation 61, we have

$$Q1(s', a_c) + \delta - Q1(s', a_k) \geq Q1(s', a^*(s')) - Q1(s', a_k) > \epsilon, \forall s' \in \mathcal{S}_s$$

Because $\delta \leq \epsilon/2$ by assumption, $\epsilon - \delta \geq \delta$. Thus

$$Q1(s', a_c) - Q1(s', a_k) > \delta, \forall s' \in \mathcal{S}_s$$

Applying Lemma 5.3 we have

$$Q(s, a_c) - Q(s, a_k) = \sum_{s' \in \mathcal{S}_s} c_{s'}(Q1(s', a_c) - Q1(s', a_k)) > \sum_{s' \in \mathcal{S}_s} c_{s'}(\delta) = \delta$$

Hence

$$Q(s, a_c) - Q(s, a_k) > \delta$$

and, by the definition of $\hat{a}$,

$$Q(s, \hat{a}) - Q(s, a_k) \geq Q(s, a_c) - Q(s, a_k) > \delta$$

But this contradicts our assumption that $a_k \in \text{pref}_\delta(s)$. Therefore our supposition that $a_k \notin \text{pref1}_\epsilon$ must be false, and we have proved that

$$\text{pref}_\delta(s) \subseteq \text{pref1}_\epsilon(s)$$

This completes the proof of the theorem for the case where $\delta \leq \epsilon/2$.

Case 2: $\epsilon/2 < \delta \leq \epsilon$

For the case where $\delta > \epsilon/2$ our compatibility criteria require that the same actions lead to maximum reward for all states in $\mathcal{S}_s$ (Equation 46). Thus

$$\text{pref1}_0(s) = \text{pref1}_0(s') = \{a : Q1(s', a) = Q1(s', a^*(s'))\}, \forall s' \in \mathcal{S}_s$$

Consequently, $\hat{a} \in \text{pref1}_0(s'), \forall s' \in \mathcal{S}_s$. If not, we would have $\hat{a} \notin \text{pref1}_0(s)$ and we could choose some other action, $a_j \in \text{pref1}_0(s)$. Because all the states share the same pref1_0 set, a_j would be in this set for each of the states, and $\hat{a}$ would not be in this set for any state. Thus

$$Q1(s', a_j) > Q1(s', \hat{a}), \forall s' \in \mathcal{S}_s \tag{62}$$

Then, by Lemma 5.3, we could write

$$Q(s, a_j) - Q(s, \hat{a}) = \sum_{s' \in \mathcal{S}_s} c_{s'}(Q1(s', a_j) - Q1(s, \hat{a}))$$

and, applying Equation 62, we have

$$Q(s, a_j) - Q(s, \hat{a}) > 0$$

But this contradicts the definition of $\hat{a}$. Thus, for any $s' \in \mathcal{S}_s$,

$$\hat{a} \in \text{pref1}_0(s')$$

and

$$Q1(s', \hat{a}) = Q1(s', a^*(s')) \tag{63}$$

Let us establish the fact that $|V1(s) - V(s)| \leq \epsilon$. By Lemma 5.2

$$V(s) = Q(s, \hat{a}) = \sum_{s' \in \mathcal{S}_s} c_{s'} Q1(s', \hat{a})$$

As we have seen (Equation 63),

$$Q1(s', \hat{a}) = Q1(s', a^*(s')) = V1(s')$$

Thus

$$V(s) = \sum_{s' \in \mathcal{S}_s} c_{s'} V1(s')$$

This means that $V(s)$ is a convex combination of the terms $V1(s')$ for compatible states s'. $V1(s)$ is trivially a convex combination of the same terms $V1(s')$. Therefore, their difference can be no greater than the difference between the maximum and minimum of those terms (Lemma 5.7). Thus

$$|V1(s) - V(s)| \leq V1_{\max} - V1_{\min}$$

where $V1_{\min} = \min_{s' \in \mathcal{S}_s} V1(s')$ and $V1_{\max} = \max_{s' \in \mathcal{S}_s} V1(s')$. Since we know, by the compatibility of values for $s' \in \mathcal{S}_s$ (Equation 44),

$$V1_{\max} - V1_{\min} \leq \delta \leq \epsilon$$

we have proved that

$$|V1(s) - V(s)| \leq \epsilon$$

Finally, we need to show that $\text{pref}_\delta(s) \subseteq \text{pref1}_\epsilon(s)$. First, note that

$$\hat{a} \in \text{pref1}_\epsilon(s)$$

according to Equation 63. Now consider any other action, say, a_k, for which

$$a_k \in \text{pref}_\delta(s)$$

This implies

$$Q(s, \hat{a}) - Q(s, a_k) \leq \delta \tag{64}$$

By Lemma 5.3 we may write

$$Q(s, \hat{a}) - Q(s, a_k) = \sum_{s' \in \mathcal{S}_s} c_{s'}(Q1(s', \hat{a}) - Q1(s', a_k))$$

But if $a_k \notin \text{pref1}_\epsilon(s)$ then $a_k \notin \text{pref1}_\epsilon(s'), \forall s' \in \mathcal{S}_s$ (Equation 43), and

$$Q1(s', a^*(s')) - Q1(s', a_k) > \epsilon, \forall s' \in \mathcal{S}_s$$

Since $Q1(s', \hat{a}) = Q1(s', a^*(s'))$ (Equation 63), we can rewrite this as

$$Q1(s', \hat{a}) - Q1(s', a_k) > \epsilon, \forall s' \in \mathcal{S}_s$$

Therefore

$$Q(s, \hat{a}) - Q(s, a_k) = \sum_{s' \in \mathcal{S}_s} c_{s'}(Q1(s', \hat{a}) - Q1(s', a_k)) > \sum_{s' \in \mathcal{S}_s} c_{s'}(\epsilon) = \epsilon \geq \delta$$

or

$$Q(s, \hat{a}) - Q(s, a_k) > \delta$$

contradicting Equation 64. Therefore our supposition that $a_k \notin \text{pref1}_\epsilon(s)$ is false, and we have established that

$$\text{pref}_\delta(s) \subseteq \text{pref1}_\epsilon(s)$$

This completes the proof.

Making the assumption that the same action is best in any of the compatible states (Equation 46) makes the proof much easier. But this assumption may be difficult to guarantee in real-world systems, for which small amounts of noise could change the balance between the values of actions which appear nearly optimal. For this reason, the result for $\delta \leq \epsilon/2$ is likely to be more useful.

5.3 Discussion

5.3.1 Necessary and sufficient conditions

McCallum (1995) develops a candidate set of criteria for state compatibility. He remarks that it would be very interesting to be able to show that these criteria are necessary and sufficient conditions for learning the task, but that he knows of no such proof. As we will see, necessary conditions are difficult to state for the general case. McCallum's criteria ensured that the action values all obeyed the Markov property, but we have seen that this is not necessary for making sound decisions, since some action values play no role in determining either the value of the state or its preferred action set. In addition, McCallum's criteria are difficult to fit to our general system model because they require separate monitoring of the action values for each path into a state—whereas this dissertation assumes that the paths are not stored explicitly, and that action values

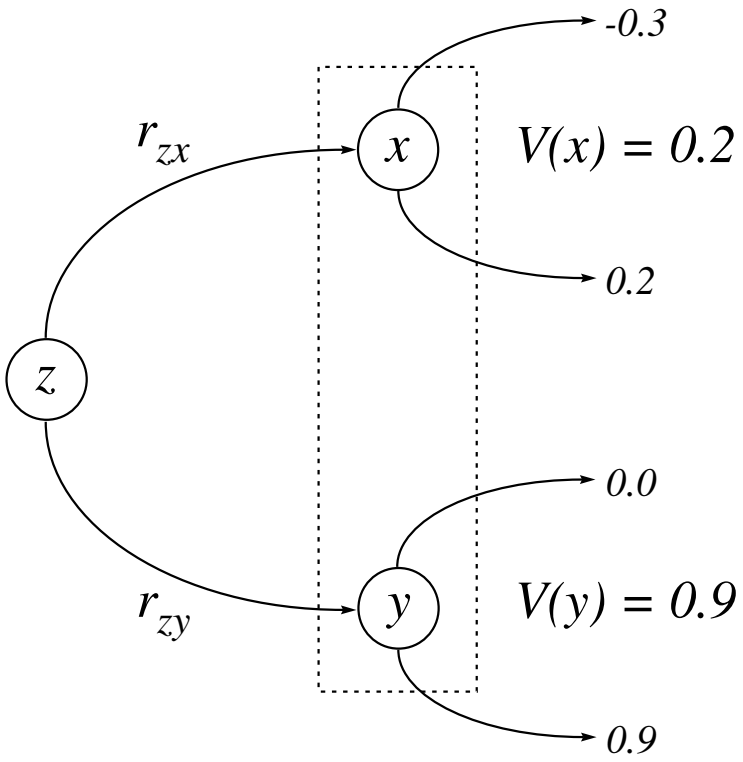


Figure 31: A value distinction which may or may not be necessary, depending on the values of r_{zx} and r_{zy}

are computed from the generalized action values, $\hat{w}_{ij}$.

Figure 31 illustrates some of the difficulties involved in trying to specify *necessary* conditions for learning the task. It depicts a situation where there is a value difference between two states, but where this difference might not matter, depending on the immediate rewards given for the actions which lead to those states.

The states x and y have the same preferred policy, so that we might be tempted to group them together. Note that if learning were not a requirement for our agent, this would be perfectly acceptable, since we would only need to categorize the states by their required actions, $\pi^*(s)$. In that case we could just *define* the correct action at z, so that the agent chooses the correct path. But *learning* a correct policy requires more

information. *If the immediate returns r_{zx} and r_{zy} are equal,* the agent needs to know that y is a much more desirable destination than x, if it is to learn the best response at z. This requires that the agent be able to distinguish x and y.

But the difference in expected reward between $z \rightarrow x$ and $z \rightarrow y$ depends partly on the immediate rewards, r_{zx} and r_{zy}. These immediate rewards could possibly cancel the differences in expected value of x and y. What this means is that, in the worst case, we cannot verify that the distinction between x and y is *necessary* without knowing the rewards for all the paths leading to x and y. This is why McCallum's criteria require the rewards to be monitored separately for each path leading to the region which includes x and y. In the case of our general system model, this information is not available, because the states are only seen as patterns of activation across a group of feature detectors, and we have no way of storing separate reward values for these paths. Even if the representation is a partition, so that there is only a single active detector, we have no way of even knowing when we have seen all possible paths leading to the region containing x and y. Therefore, we are unable to determine whether the value distinction is necessary in the general case. For now, we will be content to simply make value distinctions between states without worrying about whether they are really necessary. This may cause the agent to make a few extra distinctions, but will not cause it to lose any reward. The number of extra distinctions is minimized by the fact that the value of a state does not depend on *all* the action values (as the Markov property does), but only on the expected returns for the *preferred* actions from that state.

In this chapter, we proved that our low-level state compatibility criteria are *sufficient* conditions for meeting the ϵ-adequate representation criterion, which is our definition of what it means to have learned the task.

5.3.2 The case for partition representations

Detectors which generalize over incompatible states may obscure information needed for the agent to make sound decisions. Chapter 3 showed that even if the agent is able to learn to make sound decisions, generalizing over mixed sets tends to lower the importance of the resulting detectors. In order to maximize feature importance, each detector f_i should cover a pure set of states, so that all the states in $\mathcal{S}_{f_i}$ are compatible. One way of accomplishing this is to require a strict separation between incompatible states—like the separation between different regions of states in a partition.

Although strict separation made the analysis much easier, this is a strong requirement. I would like to see this requirement relaxed in future work, and the analysis carried out for the case where multiple, continuous-valued detectors may be active at any state. In the present version, each of the detectors which is active for s ends up being compatible with s—and therefore, with the other detectors for s. In this scenario, we could just use a single detector for this group of states.

If feature detectors are allowed to generalize (to some extent) over incompatible states, the representation may reap some of the advantages of *coarse coding* (Hinton, 1984), whereby a small number of detectors can represent a large number of different objects with great robustness. The values for a particular state will be a weighted sum of the contributions from different detectors, and the errors in these contributions will often cancel out. Coarse-coding thus trades detector importance for robustness to representational error by the detectors.

One way to manage this trade-off is to begin with a coarse-coded representation, and to tune the detectors so that their recognition sets become more pure as the agent learns. After sufficient experience, each detector would respond to a class of states

which are all compatible. In this way, the agent may be less sensitive to errors in its original representation, but may come to find features which each describe a set of states which are "similar" with respect to the task. It may still be true that a given class of states will come to be represented by a group of compatible detectors. This result could look like a partitioned representation, although one in which each partition region is represented by a set of compatible detectors—like having a single active detector f_i, where f_i is composed of several different parts. But there would again be a strict separation between detectors which represent incompatible states.

Note that, in principle, there is always an adequate partition representation for the task, since the discrete representation is always ϵ-adequate. If there are any irrelevant distinctions in the discrete representation, removing them results in a reduced partition representation which is still ϵ-adequate, but with improved cognitive economy. Therefore, we know that there exists an ϵ-adequate partition representation somewhere between the extremes of the discrete representation and the representation which makes no distinctions at all. We may conjecture that one of the purposes of feature extraction by intelligent agents is to provide a re-coding of the raw features given to the agent, in order to produce a representation with more cognitive economy. Most often this re-coding will result in a more localist representation for the particular task at hand. After all, if the representation contained a single feature detector which was uniquely active for each required action of the task, the task would be trivial!

The state compatibility criteria allow us to restrict the sharing of action values to states which are compatible, in order to avoid the harmful effects of state generalization over incompatible states. These low-level criteria provide a domain-independent way of determining *relevant* classifications, distinctions and features, according to the agent's

on-line observations of the results of its actions.

5.4 Appendix: Some Helpful Properties of Convex Combinations

Definition 5.13 (Convex combinations) *Let y be a convex combination of terms $x_1, \ldots, x_n$. Then there exist coefficients a_i such that:*

$$y = \sum_i a_i x_i$$

where

$$\sum_i a_i = 1 \text{ and } a_i \geq 0, \forall i$$

Lemma 5.4 *Let y be a convex combination of terms $x_1, \ldots, x_n$. Then y is bounded by $x_{\min} = \min_i\{x_i\}$ and $x_{\max} = \max_i\{x_i\}$. That is,*

$$y \in [x_{\min}, x_{\max}] \tag{65}$$

Proof: We know that

$$x_{\min} \leq x_i \leq x_{\max}$$

Since y is a convex combination of the x_i, there exist coefficients a_i which allow us to express y as the sum $\sum_i a_i x_i$, where $\sum_i a_i = 1$, according to the definition of convex combinations.

$$\sum_i a_i x_{\min} \leq \sum_i a_i x_i \leq \sum_i a_i x_{\max}$$

Thus

$$x_{\min} \sum_i a_i \leq y \leq x_{\max} \sum_i a_i$$

and

$$x_{\min} \leq y \leq x_{\max}$$

Lemma 5.5 *If z is a convex combination of terms $y_1, \ldots, y_n$ and each y_i is itself a convex combination of terms $x_1, \ldots, x_m$, then z is a convex combination of the terms $x_1, \ldots, x_m$.*

Proof: Since z is a convex combination,

$$\exists a_i \geq 0 : z = \sum_i a_i y_i \text{ and } \sum_i a_i = 1$$

Because y_i is a convex combination of the x_j,

$$\exists b_j^i \geq 0 : y_i = \sum_j b_j^i x_j \text{ and } \sum_j b_j^i = 1$$

Thus

$$z = \sum_i a_i \sum_j b_j^i x_j = \sum_i \sum_j a_i b_j^i x_j = \sum_j \sum_i a_i b_j^i x_j = \sum_j x_j \sum_i a_i b_j^i$$

Therefore

$$z = \sum_j c_j x_j \text{ for } c_j = \sum_i a_i b_j^i$$

Since c_j is the sum of products of non-negative coefficients, $c_j \geq 0$. In addition,

$$\sum_j c_j = \sum_j \sum_i a_i b_j^i = \sum_i \sum_j a_i b_j^i = \sum_i a_i \sum_j b_j^i = \sum_i a_i = 1$$

Therefore, z is a convex combination of the x_j.

The next lemma shows that if two numbers both lie within a particular segment of the real line, then the difference of those numbers cannot be greater than the length of that segment.

Lemma 5.6 *Let $y \in [x_{\min}, x_{\max}]$ and $z \in [x_{\min}, x_{\max}]$. Then*

$$|y - z| \leq x_{\max} - x_{\min}$$

Proof: We know that

$$x_{\min} \leq y \leq x_{\max}$$

and

$$x_{\min} \leq z \leq x_{\max}$$

Let $u_1 = \min\{y, z\}$, and $u_2 = \max\{y, z\}$. Then

$$|y - z| = u_2 - u_1 \leq x_{\max} - u_1 \leq x_{\max} - x_{\min}$$

Lemma 5.7 *If y and z are both convex combinations of terms $x_1, \ldots, x_n$ then*

$$|y - z| \leq x_{\max} - x_{\min}$$

Proof: Because y and z are both convex combinations of the x_i, we know by Lemma 5.4 that $y \in [x_{\min}, x_{\max}]$ and $z \in [x_{\min}, x_{\max}]$. Therefore, we may apply Lemma 5.6.

Chapter 6

Case Studies in On-Line Feature Extraction

> Simplicity is difficult, not easy. Beauty is simple. All unnecessary elements are removed, only essence remains.
>
> —*Alan Hovhaness*

6.1 Introduction

The previous chapters developed a framework for understanding the role of representation in reinforcement learning. They showed how the idea of cognitive economy may be expressed in terms of objective criteria based on the action values in a task. This chapter demonstrates how such criteria may be implemented and applied by a system which constructs its representation as it goes about learning the task. The two tasks considered are the puck-on-a-hill task, and the pole-balancing task.

Chapter 3 worked out criteria for representational adequacy (the ϵ-adequacy criterion) and state compatibility. This chapter makes a straight-forward application of these ideas to the case where the agent's representation is a partition of the state-space. The ϵ-adequacy criterion provides a basis for deciding whether the representation is adequate to properly classify the current state-action-reinforcement

experience. The state compatibility criteria provide the agent with a test for determining when it is safe to group two states together in the same region. When the adequacy test indicates a "surprising" state, the agent takes this result as a cue to look more carefully at that part of the task; it uses the compatibility test to compare the surprising state with some representative state for that region, in order to determine whether the region should be split.

6.1.1 Making Relevant Distinctions

The system described here attempts to group states together unless they must be separated in order to make distinctions which are necessary in the task: policy distinctions and value distinctions. Although it makes distinctions which prove important in the task, this is a different approach than the one taken in my earlier work on *importance-based feature extraction* (Finton and Hu, 1994 and 1995). That work monitored the importance of the features and tuned feature detectors in ways which increased a measure of importance. Instead of attempting to increase feature importance directly, the current work attempts to indirectly increase the importance of feature detectors by preventing action values from being generalized over incompatible states. Another difference is that the earlier studies tuned a fixed number of detectors, while the system presented here generates new feature detectors as needed.

This is a unique approach to finding relevant distinctions, in that it ignores errors in action values unless they make a difference in either the agent's policy or the overall value of a state. While this approach could be used to construct a comprehensive map of the state-space, it would only do so if the task's reinforcements made the higher level of detail relevant. Even in that case, other methods of feature extraction might

be preferable (for example: Kohonen's Self-Organizing Map, 1990; McCallum's U-Tree, 1995), since this approach is based on the assumption that not all of the details are relevant to the agent, so that it is possible to find adequate representations with greater cognitive economy than that given by a complete state-space mapping.

The price we pay for increased cognitive economy is the need for additional information while constructing the representation. To make state compatibility assessments, we need action value profiles for the states in question, providing estimates of the complete set of action values, instead of just the value of a single action. We need complete profiles because leaving out one of the actions might cause us to get the wrong policy for a state, or give a bad estimate of the state's value if the left-out action happened to be the best one. The current system handles this by a form of active learning, in which the agent may choose the starting states of its experiments as well as the actions taken. Although this is a straight-forward application of the ideas, the same benefits could be realized by other means than active learning, as discussed later in the chapter. As long as the agent has access to action value profiles for its states, it can make the needed compatibility assessments.

These compatibility assessments compare the action value profile of the current "surprising" state with that of a representative state from the same state-space category. If the two profiles are sufficiently different, the states are incompatible. What is novel about this approach is that the two profiles can differ greatly on one or more of the action values without the states being considered incompatible, as long as that difference does not affect the policies or the maximum values obtainable from those states. This test is based on the assumption that cognitive economy has to do with representing the world only well enough for the agent to always choose the correct

action. Accurate action values help, but are not an end in themselves.

6.1.2 Methodology

If this cognitive economy approach allows us to distinguish relevant features, it should lead to algorithms which can automatically construct high quality representations for the agent's task. One test of this approach is simply to evaluate the performance of a system which uses these ideas to construct its representation for the task as it goes about learning the task. Although this technique is often seen in the literature, good performance in the task does not necessarily indicate a high quality representation; evaluating a representation is different from simply evaluating performance.

Several other criteria may help evaluate the quality of the representation. If the representation contains a small number of features, that may be taken as evidence of cognitive economy, provided that those features allow the agent to make the necessary distinctions in its task. If the representation is reusable by other agents which learn the task, that is evidence that it captures important features of the task, rather than artifacts of a particular training regimen. If the representation allows good performance from a variety of starting points, that may indicate a high level of quality throughout the relevant parts of the state-space. Therefore, our evaluation methodology should test the representation independently of the system which produced it, and should exercise the representation over a significant portion of the state-space. Our quality assessment will be based on the number of features and the effectiveness of the representation in the task.

The effectiveness of the representation may be shown most effectively by a learning

curve plotting the performance of a test system using that representation; the performance is plotted as a function of training time for learning the action values. Learning curves are especially useful for evaluating representations, since a single measurement of either learning speed or performance is likely to favor either small representations, which tend to learn quickly, or larger representations, which often allow a higher standard of performance to be learned. In addition, learning curves show whether a representation reliably supports good performance, or only results in occasional successes. If a learning curve is produced, it should be averaged from multiple experimental runs, in order to minimize system initialization effects.

The experiments reported here consisted of two stages: a representation generation stage and a representation testing stage. In the generation stage, a learning system applies an algorithm based on cognitive economy to construct a representation for the task it is learning. The output of this stage is the specification of a new state-space representation for the task. If the algorithm is successful, it will not only learn to perform well in the task, but it will produce a representation which captures the important features of the problem, in a form reusable by agents that do not learn the representation.

In the testing stage, we take a reinforcement learning system with a fixed representation, and replace that representation with the new one produced by the system in stage one. We keep that representation frozen and run a battery of tests on this system to measure its performance. Since the tester is separate from the agent which produced the representation, it is able to evaluate different representations fairly. Although the tester does not modify the representation it tests, it must still learn the action values for the task, which is why it improves with training.

To produce learning curves, the tester generates a series of performance scores, with each score based on the performance of the system in a batch of trials with learning turned off. By alternating periods of training with performance measurements, the tester generates a plot of performance versus training time. These plots allow us to compare the quality of different representations.

The number of learning trials between measurements depends on the lengths of the trials. The system will not stop a running trial, since the tasks studied here have reinforcement only at the ends of trials. The system interrupts its training with a testing session when it reaches either of a preset number of learning steps or a preset number of learning trials. The two-part test ensures that the system generates enough data points both early (when trials are short, so the number of trials dominates) or late (when trials are long, and the number of learning steps dominates) in the series. Training and testing continue until the agent has trained for a specified minimum number of steps. The system produces a final, averaged plot of performance versus training time by averaging a series of learning curves generated by this protocol, each curve resulting from a different random number initialization.

The tasks presented in this chapter—puck-on-a-hill and pole-balancing—have often served as reinforcement learning benchmarks in the literature. Examples include Michie and Chambers (1968), Barto, Sutton, and Anderson (1983), Sutton (1984), Anderson (1986), Finton and Hu (1994), Yendo Hu (1996), Hu and Fellman (1996), and Likas (2001). Some of these studies produced results in the form of learning curves, and some initialized trials at randomly-chosen starting points instead of always starting at the equilibrium point; but the method introduced here for evaluating learned

representations in a separate system appears to be unique. This method allows different representations to be compared separately from other aspects of the reinforcement learning problem.

The random starting states were chosen as follows. For each task, an earlier series of experiments yielded a set of extreme values for the state-space coordinates; the tester's training trials were initialized with values from the central third of this observed state-space, and its test trials were initialized to points from a slightly smaller zone—the central quarter of the space. Initializing trials to random states ensures that the representation is tested over a significant portion of the space, providing a more meaningful indication of the quality of the representation.

6.2 An Algorithm for On-Line Feature Extraction

This section describes the learning system used in the first phase of the experiments. The system is built upon a nearest-neighbor representation of the state-space, which divides the space into a series of partition regions. These regions are built out of the Voronoi regions (set of points closest to a particular point in the space) about each of a series of prototype states given to the representation. Some regions consist of a single prototype and its Voronoi region, while others are compound regions consisting of a set of merged Voronoi regions. The compound regions are represented by a primary prototype state; this state is taken as the representative state for any of the states which fall in that region, even though some other state may be their nearest-neighbor prototype. If the state to be classified lies within a simple, un-merged region, its primary prototype will be the nearest-neighbor.

The system learns action values by a combination of Q-learning and an active strategy by which it examines "surprising" states at the ends of its trials. When the system's current action leads to unexpected results, it pushes the current state on a replacing-stack data structure. At the ends of trials, the system conducts mini-trials from the states on its stack. These investigations produce action-value profiles which the system uses to update its action values, and also to determine whether the representation adequately represents these surprising states. If these states are not compatible with their current classifications, the system splits the state-space region by creating a new region for the surprising state. The compatibility test compares the action-value profile for the surprising state with the profile for its primary prototype state in the representation. Region splitting is accomplished by simply adding the surprising state as a new prototype; it then becomes the primary prototype of a new Voronoi region in the space.

6.2.1 Top Level of the Algorithm

The top level of the algorithm is given in Figure 32. It outlines the function `get_action`, which takes parameters for the current state, the current reinforcement (r_t), and a flag indicating whether the current state is a terminal state.

Given: current state, current reinforcement, terminal state flag

k = region for current state
j = region for the previous state
a = index of the last action
if terminal state or `reliable_source`(k)
then update $Q(j, a)$:
    if terminal state
        $Q_{\text{new}} \leftarrow r_t$
    else
        $Q_{\text{new}} \leftarrow r_t + \gamma \max_i Q(k, i)$
    $Q(j, a) \leftarrow (1 - \alpha)Q(j, a) + \alpha Q_{\text{new}}$
    `Updates`$(j, a) \leftarrow$ `Updates`$(j, a) + 1$

if this experience was `surprising`
  or j has never been investigated
  or a weighted coin flip returns heads
then push (previous state, region j) on the stack
    for later investigation

if terminal state
    call `process_stack()` to investigate surprising states
else
    select next action based on current policy and $Q(k)$

Figure 32: Top level of the algorithm.

During the trials, the weight updates are those of Q-learning, except that no update is done when the current experience is not considered reliable. For the feature extraction ideas to have a chance to succeed, the system must produce accurate action-value profiles. The function

`reliable_source()` determines whether the system has had enough experience with the state in question to trust its action values as reliable for backing up to other states. To do this, `reliable_source()` consults `Updates`(j, a), which is incremented whenever $Q(j, a)$ is updated. A state is considered a reliable source for backing up values if one or more of its actions has been updated at least `MIN_UPDATES` times. The standards are higher for feature extraction decisions: the function `reliable_prototype()` demands that every action value have been updated `MIN_UPDATES` times, instead of just one action. Reinforcement from terminal states is always considered reliable.

The looser reliability criterion for value updates allows the values to begin to be updated as soon as the agent experiences reinforcements. If we demanded the stronger criterion for the initial backups, cyclic tasks such as the pole and puck tasks would experience deadlock conditions which would prevent some states from ever being updated at all. The looser criterion still allows the system to avoid some of the bogus value updates it would otherwise make early in the run, when the action values are not grounded in the task reinforcements. The value of `MIN_UPDATES` used in these experiments was 3.

6.2.2 Recognizing Surprising States

The function `surprising()` focuses attention on parts of the state-space which appear to be inadequately learned or categorized. It flags a state-action pair as "surprising" if it appears to violate the ϵ-adequacy criterion developed in Chapter 3. Given the

outcome of an action—its immediate reward and the value of the resulting state—`surprising()` determines whether the outcome is sufficiently surprising to call into doubt the system's estimate of the value of the state or the preferred policy at that state.

The ϵ-adequacy criterion compares the action-value profile of a region with the profile obtained for a particular state in that region. Unfortunately, the agent's current experience only provides the current result for a single action, instead of the complete profile for a state. Therefore, `surprising()` compromises by simply checking to see if updating the value for the observed action appears to change the region's value or its policy. If so, `surprising()` flags this state as worthy of further investigation.

The key ideas are these: (1) Each state has a set of preferred actions, and bad state generalization can cause the agent to select the wrong actions if the action values for the region do not reflect the correct policy for one of the states in that region. We may tolerate sloppy generalization that limits the number of good actions we take from a particular state, but we cannot allow the kind of bad state generalization that causes us to choose poor actions. (2) If one of the states results in a much better (or worse) outcome than the rest of its region, the agent may look at the values averaged over the region and be misled whenever it happens to be in that particular state. Figure 33 outlines the criterion computed by `surprising()`.

This criterion flags a state if the observed information indicates that this state ought to be distinguished from its region on the basis of either its overall value or its policy. The value distinction can be seen by comparing the old value (for the region) with the new value (modified by information observed for this particular state). The criterion makes an action distinction if the current policy for the region (given by pref_δ) contains

`surprising()`:

Given: previous region (j), previous action (a)
resulting region (k), resulting reinforcement (r)
terminal state flag
$V \leftarrow$ maximum action value for region j
$\text{pref}_\delta \leftarrow$ set of preferred actions for region j
(these have action values within δ of V)

Compute new action values:
if terminal state
$Q_{\text{new}}(a) \leftarrow r$
else
$Q_{\text{new}}(a) \leftarrow r + \gamma$ (greatest action value from region k)
For actions $i \neq a$, the new values are the same as the old ones:
$Q_{\text{new}}(i) \leftarrow Q(j, i)$

$V1 \leftarrow$ maximum action value, using Q_{new}
$\text{pref1}_\epsilon \leftarrow$ set of preferred actions, using Q_{new}
(these have action values within ϵ of $V1$)

Flag this state if:
$|V1 - V| > \epsilon$
or
Some actions in pref_δ are not in pref1_ϵ

Figure 33: Selecting "surprising" states for further investigation.

actions which appear to be bad ones for this particular state (because they are not in pref1_ϵ). The tolerance δ describes how close a value has to be to the optimal value in order for the agent to select that action under a greedy policy. The tolerance ϵ describes the maximum amount of error (called *incremental regret* in Chapter 3) we are willing to accept for any single action.

6.2.3 Investigating Surprising States

After every trial, the system calls `process_stack()` to investigate states which it found "surprising." Because the stack is a last-in-first-out data structure, this causes the system to explore states which occur at the ends of episodes before it explores earlier states. In a non-cyclic task, this property allows the system to focus on the frontier of learned states, where the action values have been grounded in the reinforcement given by the environment. Actions leading to terminal states are learned first, then the action values of states one step earlier. As the system learns about states near the ends of trials, they stop being surprising, and the system focuses its attention on states which precede those states. In this way, the action values are learned from the end states backwards to the beginning states, but without all the action-value backups from internal states whose values have not yet been learned.

Because the stack is implemented as a replacing-stack, pushing any item causes the stack to remove any previous instances of that item before adding the new item. To enable the system to cope with continuous state-spaces, in which the same exact state might never be repeated, the stack regards states from the same region as "the same." Therefore pushing a state removes any other states having the same region from the stack. This ensures that the stack size does not grow without bound: the number of items on the stack is limited by the number of state-space regions in the representation, and a particular region will not be explored more than once for any session. These are desirable qualities for cyclic tasks like those presented in this chapter, because a single region might otherwise fill the stack with states seen during repeated passes through the region.

The system investigates a state by conducting mini-trials for each possible action

from that state. These trials only last long enough for the agent's state to enter another region or for it to reach a terminal state. In the case of an action which leads back to the same region, the investigation times out after a certain number of steps. If a single action takes the agent out of the region, the mini-trial stops after one action.

Feature extraction is not performed after every trial. If this investigation is not one in which the system will perform feature extraction, the system applies the results of its investigation by updating action values according to the new profile. Like the Q-learning updates done in the top level of the algorithm, these updates only back up values when the resulting states (from the investigations, in this case) are determined to be reliable. Unlike those updates, the learning rate for the active updates decreases with the number of times the action value has been updated, until it hits a specified minimum value. Figure 34 summarizes the active investigation scheme.

```
process_stack():
    while not empty( replacing stack )
        s ← pop( replacing stack )
        j ← region for s
        if feature extraction waiting interval has passed
            update_representation(s, j)
        else
            get profile and reliabilities from investigate(s)
            for each action a
                if reliabilities(a) = yes
                    increment Updates(j, a)
                    update Q(j, a) according to profile(a), and
                                    learning rate  alpha(j, a)

investigate(s):
    for each action a
        start a mini-trial at s
        keep executing action a until: region exit
                                       or terminal state
                                       or time-out
        s' ← resulting state from mini-trial
        profile(a) ← resulting value from mini-trial
        reliabilities(a) ← reliable_source(s')

alpha(j, a):
    if Updates(j, a) ≤ ENOUGH_SAMPLES
        return 1.0/Updates(j, a)
    else
        return 1.0/ENOUGH_SAMPLES
```

Figure 34: Strategy for Active Investigations of Surprising States.

6.2.4 Adding and Merging State-Space Regions

When the system performs investigations on its stack of surprising states, it only performs feature extraction at certain intervals, when the trial number is a multiple of the preset value `FE_INTERVAL`. This allows the action values time to settle between changes in the representation. Figure 34 has already shown the basic strategy for the case when feature extraction is not done. When feature extraction is done, the system begins by finding the nearest-neighbor prototype for the current state. If the nearest-neighbor is not the primary prototype for the region, the system may need to test the current state's compatibility with both the primary prototype and the nearest-neighbor prototype. When necessary, the system creates new regions by adding new prototypes, each one defining a new Voronoi region of the state-space. Regions may be merged, resulting in a compound region whose primary prototype is the prototype with lowest index.

If the current information is not reliable or the current state is compatible with its primary prototype, the representation is left alone and the system updates the current region's values normally, according to the new profiles for the state and its primary prototype.

If reliable information says that the current state is not compatible with its primary prototype, the system needs to find a new category for the current state. This usually leads to a decision to add the current state as a new prototype. The exception occurs when the region has a nearest-neighbor prototype which is itself incompatible with the primary prototype; in this case, the system detaches the nearest-neighbor prototype from the primary prototype, and makes it the primary prototype of its own region. If it does so and the current state is compatible with the nearest-neighbor prototype, the system can avoid adding the current state as another new prototype.

After any necessary changes are made to the representation, the system makes use of its new profile information to update values for the appropriate regions. As a final step, the system executes a state consolidation procedure, although it only does so once for every `CONSOLIDATE_TRIGGER` executions of its feature extraction routine. When consolidating states, the system looks at every pair of primary prototypes (j, k). If the information passes the test of `reliable_prototype` and the two states are compatible, the system merges them. Figure 35 summarizes the feature extraction system.

```
update_representation(s, j):

    Given: s = current state
           j = current region
           (trial number is a multiple of FE_INTERVAL)

    s_p ← primary prototype for s
    s_p2 ← nearest-neighbor prototype for s, if it exists
    get profiles and reliabilities for s and s_p from investigate()
    update values for j by the profile for s_p
    if should_split(s, s_p) = no or reliable_prototype(j) = no
        update values for j by the profile for s
    else
        reduce reliability info for j, because it will be split
        get profile and reliabilities for s_p2, if it exists
        add s as a new prototype, unless should_split(s_p, s_p2)
                                     and should_split(s, s_p2) = no
        if should_split(s_p, s_p2)
            detach s_p2 from s_p
            if we did not add s as a prototype
                update detached region with profile for s
    if consolidate waiting interval has passed
        consolidate compatible states

should_split(s_1, s_2) :

    if all actions are reliable from s_1,
      all actions are reliable from s_2,
      compatible(s_1, s_2) = no
    then
        return yes
    else
        return no
```

Figure 35: Feature Extraction Algorithm.

```
compatible(s1, s2):

Given: profile1 for state s1
       profile2 for state s2
       ε = maximum error tolerance for wrong choices

pref1_0(s) ← set of best actions for state s
pref1_ε(s) ← set of actions within ε of best for s
V1(s) ← action value for best action from s

if |V1(s1) − V1(s2)| > ε
  or pref1_0(s1) ≠ pref1_0(s2)
  or pref1_ε(s1) ≠ pref1_ε(s2)
then
     return no
else
     return yes
```

Figure 36: Judging the Compatibility of Two States.

Figure 36 outlines the state compatibility test, which is based on the criteria developed in Chapter 3, taking $\delta = \epsilon$ (which is allowed by the criteria, and seems to result in fewer irrelevant states being generated). The test determines whether two states should be members of the same state-space region, so that we can always distinguish states which either require different action policies (a policy distinction) or which lead to significantly different outcomes (a value distinction). Policy distinctions enable the agent to choose actions wisely at this point in the task, since it would choose bad actions if it could not distinguish states requiring different policies. Value distinctions allow the agent to choose good actions from earlier states; for example, if the current state is especially good, the agent needs to distinguish it from other states if it is to be able to learn to reach this state from an earlier one.

6.2.5 Further Considerations

The system presented here is a first application of the ideas worked out in previous chapters, and it makes several assumptions which do not hold for all reinforcement learning tasks. First, the active system explores its state-space at the ends of trials, which presupposes that the agent's task is episodic. But we can often make a task episodic by choosing certain states as terminal states. A more serious limitation is that some real-world tasks cannot allow the controller to reset the system state at will (although this is no problem for any task which is solved through simulation). As discussed below, active investigations are only one way of gaining state profiles. The system also depends on the task not being too stochastic. Otherwise, a "surprising" criterion would need to be more sophisticated than the one presented here, taking into account trends and averages over very many instances. The decreasing learning rate used in the active investigations helps somewhat, since the learning rate $1/\texttt{Updates}(j, a)$ causes the updated value to be the average of all the instances seen. This is important when the state-space regions are too coarse, since the action values may appear stochastic even in a deterministic task, simply because they really belong to different kinds of states which get updated together.

I chose a nearest-neighbor partitioning of the state-space for several reasons: it allows the system to easily adapt its partitioning to regions with irregular shapes, the system may split regions by simply adding new prototype states, and when we need a representative state for a region in order to make a compatibility assessment, the prototype state is at the geometric center of its region in the state-space. One drawback is the difficulty of deciding which regions are neighbors when consolidating regions; the current system merges any two compatible regions without considering

whether they are neighbors. Also, although the present implementation seems quite efficient, the nearest-neighbor scheme requires a more sophisticated implementation to scale to large tasks.

The active strategy was a natural choice for a method of grabbing the action-value profile for a state, since we need the values of all the actions. The main reason for getting all the values at once is that the values are changing as the agent is learning the task. Even if two actions lead to the same resulting state, the agent might mistakenly believe them to have different values if the values were computed at different times. It is important to point out that the active system is still technically a form of Q-learning, since Q-learning does not specify how the value backups must be distributed among different state-action pairs—only that they continue to be sampled. By pushing randomly-chosen states onto the stack, the current system ensures that values continue to be sampled.

There are also non-active strategies for implementing these ideas. One alternative is to adopt a more stringent test for surprising states (using `reliable_prototype()` instead of `reliable_source()`); then immediately add any state thus selected as a new prototype for a region, instead of first exploring states through active investigations. In the end, one can only decide to split regions by first making tentative splits and exploring their effectiveness. The current system does this when it assesses the compatibility of a state with its primary prototype, since it is making a hypothetical separation between those two states in order to see if the region should be split. Splitting on the basis of the surprising test makes a less tentative initial separation of the states, but the algorithm's state consolidation procedure could prevent the system from accumulating unneeded regions.

Another promising approach is for the agent to save every state transition it experiences as an (s, a, r, s') tuple which is stored with the nearest-neighbor prototype state. These could be reorganized as regions are merged or detached, and the tuples could be reassigned when new prototypes are added. In this way, the agent would construct an increasingly accurate model of its world, and the agent could perform investigations by querying this model instead of by requesting that the environment reset the system state for actual trials in the world. This would also remove the episodic task limitation, since the agent could conduct investigations internally, whenever it wished. This approach has much in common with Moore and Atkeson's (1993) Prioritized Sweeping algorithm, which uses a priority queue to select states for re-examination: states whose values have significantly changed, as well as states which are predicted to lead to them. The replacing-stack performs a similar function by giving priority to states which are surprising, in order of recency. The states leading to those states are often the next states to be considered surprising, leading them to be investigated as well—unless the change in values did not make a value or policy distinction. The other difference, of course, is that Moore and Atkeson's system assumed a fixed representation, while this projected system would perform feature extraction.

One of my design goals in developing the active system was to provide accurate action values to the feature extraction routines. The active algorithm allows the system to focus on the frontier of states whose values are being changed because of the agent's current interaction with the world, or because they lead to states whose values are changing. The test for surprising states allows the agent to select training examples which are likely to be informative. The functions which prevent backups from states with low experience have the same aim of safeguarding the integrity of the values being

used to guide feature extraction.

Even so, I found that adding a Q-learning update to the top level of the algorithm improved the performance of the active system, as did allowing values to settle for a number of trials between calls to the feature extraction routines. I believe that the reason these helped was that when the representation is modified, the action values need to be revised to reflect the changes. This takes time. The top-level update results in many more backups to the values between the active investigations, and this seems to smooth out some of the errors in the values.

6.3 Initial Tests

Initial tests included a simple gridworld task like the one discussed in Chapter 1. The purpose of these tests was to verify that the system could find reasonable representations for a simple task. As expected, the system found good representations for the task, whether it started from scratch or was given a representation to refine. The system was able to consolidate irrelevant detectors in the supplied representation, and was able to complete the representation so that the task could be solved.

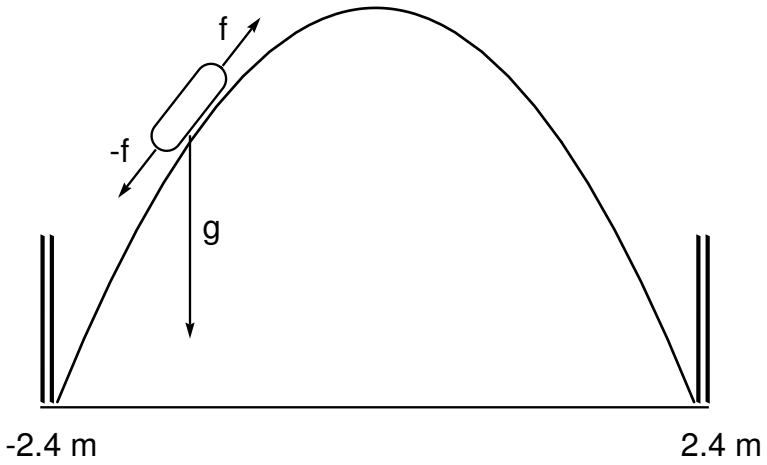
Other tests included fixed-representation experiments with the pole balancing task, designed to test the ability of the active system to learn action values. Three different controllers were tuned and tested: a Q-learning controller (with enhancements to prevent backups of "unreliable" information), a Monte Carlo controller, and the active controller (without the addition of Q-learning updates during trials). Each controller used the same 162-box representation found in (Barto, Sutton, and Anderson, 1983). Testing consisted of evaluating each controller on 10 sets of trials, each set having a

different random number seed. (The controllers select actions randomly when the values are very close). Each set continued until the system met the success criterion or 10,000 trials were run. The success criterion was that of Likas (2001): in each trial, the system state was initialized to random points in the state-space; when a trial lasted for at least 120,000 steps, the system was reset for a trial starting at the equilibrium point with learning turned off. If this second trial also reached 120,000 steps of balancing, the run was determined to be successful. Given 10 sets of trials, Monte Carlo succeeded in eight of the 10, and the enhanced Q-learning and active controllers succeeded in nine of the 10 sets. This is evidence that the active system was able to learn the action values, which are the basis for state-compatibility assessments.

6.4 Case Study: Puck-On-A-Hill Task

In the puck-on-a-hill task (Yendo Hu, 1996), the agent controls a puck which it must learn to push to the left or the right in order to maintain its position on the top of a hill. The agent's only reinforcement comes when it falls too far down the hill on either side and hits the containing wall. When that happens, the agent is given a reinforcement of -1, and the episode ends. Figure 37 illustrates the task. This task is similar to the pole-balancing task described in the next section, but with a two dimensional state-space having components for position and velocity only. Positive x represents a position on the right, while positive θ represents a position on the left. Positive f pushes the puck toward the right. The equations of motion are as follows:

$$\begin{aligned} x(t+1) &= x(t) + \Delta v(t) \\ v(t+1) &= v(t) + \Delta \frac{(f(t) - mg\sin\theta(t))\cos\theta(t)}{m} \end{aligned}$$



State	x, position of puck (meters) v, velocity of puck (meters / second)
Control	f, force on puck (Newtons)
Constraints	$-2.4 < x < 2.4$ $f = \pm 3.0$
Equation of hill	$y = -\beta x^2$
Parameters	$\beta = 0.3$ $g = 9.8$ m / s^2, gravitational acceleration $m = 1.0$ kg, mass of puck $\Delta = 0.02$ s, sampling interval

Figure 37: The puck-on-a-hill task: balance the puck on the hill to avoid negative reinforcement from hitting the wall.

$$\theta(t) \quad = \quad \arctan(-2\beta x(t))$$

6.4.1 Analysis

The puck's acceleration is determined by its thrusters and the downward pull of gravity. Near the center of the hill, the thrusters dominate the force of gravity, and the puck can push itself back to the crest of the hill as long as its prior velocity is not too large. Away from the center, the hill's slope becomes increasingly steep, so that gravity overwhelms the contribution of the puck's thrusters. Therefore, once the puck has fallen too far down the hill, it loses the ability to climb back up, and fails shortly after.

Just where this "point of no return" lies depends on the puck's velocity. From the equations of motion, we see that the acceleration on the puck is zero when

$$f(t) = mg \sin \theta(t)$$

For positive θ (the left side of the hill) this occurs at $x \doteq -0.54$; for negative θ, the acceleration is zero for $x \doteq 0.54$. If the puck is placed farther than 0.54 meters from the center of the hill and has zero velocity, it will never be able to push back up the hill. If the puck is already moving up the hill, it may be able to coast back into the central region where its thrusters can overcome the pull of gravity. Therefore, the "point of no return" lies farther down the hill when the puck has a higher initial velocity toward the center.

The agent must keep the puck within the region where its thrusters are effective in controlling the puck. We will call these states *controllable states,* and we will call the states which are past a "point-of-no-return" *doomed states.* Figure 38 shows the controllable states, which form a band which falls roughly diagonally through the middle of the state-space. This figure was produced by running puck experiments at each point

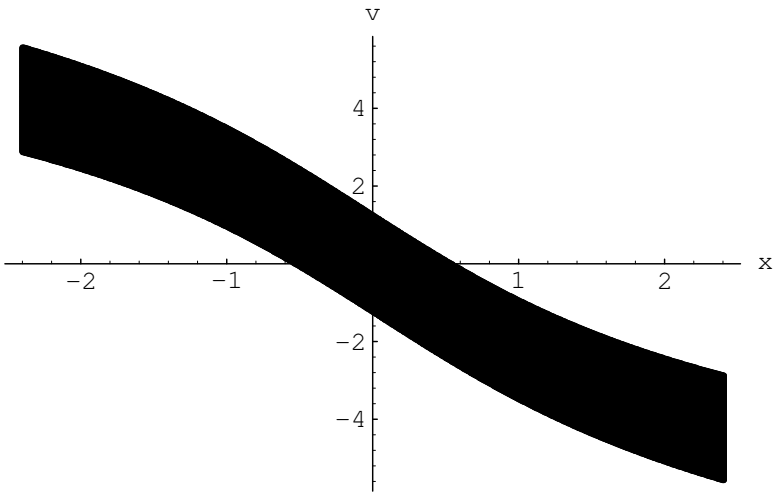


Figure 38: Controllable states: states outside this band result in failure.

of a very fine grid. (Resolution was 0.01 in both x and v). Doomed states are those from which the puck can not avoid falling to one side or the other: it either falls back down the left side of the hill while continually pushing to the right, or falls to the right while continually pushing to the left. The remaining states are the controllable states, from which the puck can avoid falling to either side.

In order to determine an optimal policy for the task, we need to determine when the puck must be pushed to the right, and when it must be pushed to the left. If pushing right from a controllable state s_c results in a doomed state, then we may classify s_c as a "must-push-left" state. Similarly, if pushing left results in a doomed state, we may classify s_c as a "must-push-right" state. If both left and right lead to other controllable

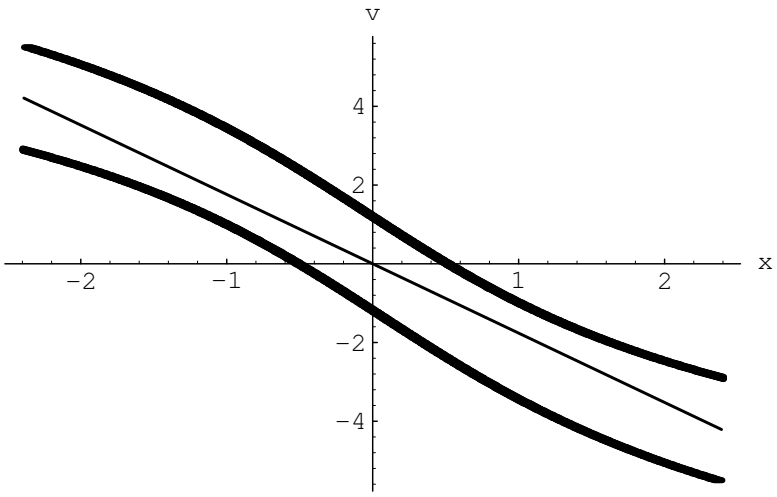


Figure 39: An ideal representation: must-push-left states (top curve) and must-push-right states (bottom curve) are separated by the diagonal line.

states, we may classify s_c as a "don't care" state. The critical states are the must-push-right and must-push-left states, where the agent's next action determines whether it succeeds in the task. These states are on the edges of the controllable zone; the states in the middle of the controllable zone are don't-care states because neither action will push the puck past the boundary of the zone. Figure 39 shows the critical states: the must-push-left states make up the top curve, and the must-push-right states make up the bottom curve. These plots were determined by testing each controllable state found in the earlier simulation, evaluating the controllability of the states which result after a single push to the left or right.

This analysis shows that the optimal policy is not to simply push toward the center

of the hill. For example, if the puck has sufficient velocity to the right, it should push to the left, even if it is already on the left side of the hill. If the velocity is too high, the puck will reach the other side of the hill, but it will not have enough time to slow down and avoid hitting the wall on that side (failure). The optimal policy is to push to the left in the must-push-left states (the top curve in Figure 39), and to push to the right in the must-push-right states (the bottom curve). Any representation which separates these two classes of states will be adequate for the task. In Figure 39, the diagonal line cleanly separates the must-push-right states from the must-push-left states. Therefore, the representation which simply splits the state-space by this line (having equation $v = -1.7615x$) is ideal for the task, because it makes the necessary distinctions and has minimal size (only two categories). This simple diagonal-split representation provides a benchmark against which we may evaluate the effectiveness of representations constructed by the learning system.

6.4.2 Results

The results compare the performance of a test system under different state-space representations. Each representation was tested by inserting it into the test system and generating a series of 10 learning curves, which were then averaged. The learning curves plot performance against the number of training steps experienced by the test system. Each performance score is the median trial length for a batch of 50 trials conducted with learning turned off. The test trials were stopped if they reached 5,000,000 steps. After 50,000 steps of training, the diagonal-split representation and the learned representation both attained averaged performance scores of 5,000,000 steps.

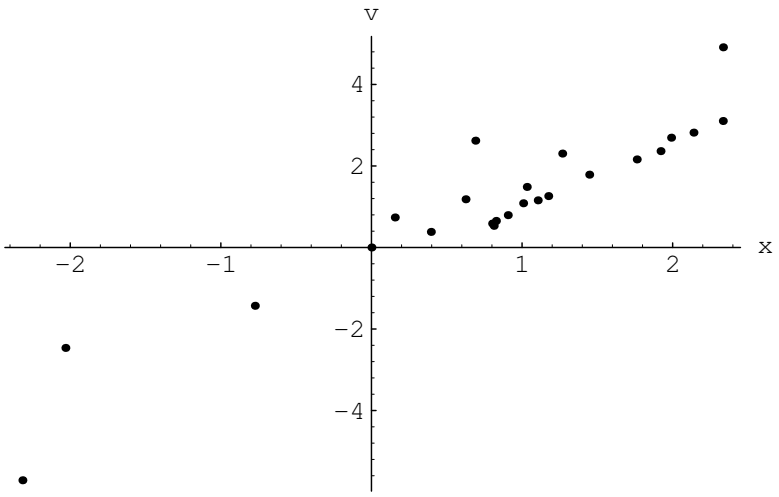


Figure 40: Representation constructed automatically, from scratch (24 categories).

Generated representations

Starting from scratch, the system generated a representation consisting of 24 prototype states, shown in Figure 40.

In another experiment, the system was seeded with a representation consisting of the two (x, v)-space points (0.2680, 0.6200) and (-0.2680, -0.6200). These points are on either side of the controllable zone; although the line connecting them is not quite perpendicular to the diagonal split of the ideal representation described earlier, these two points were thought to be sufficient to distinguish must-push-left points from must-push-right points. The objective of this second experiment was to verify that the state compatibility criteria do not lead to the generation of unnecessary states. This was confirmed by the resulting representation, which simply added two states at the usual

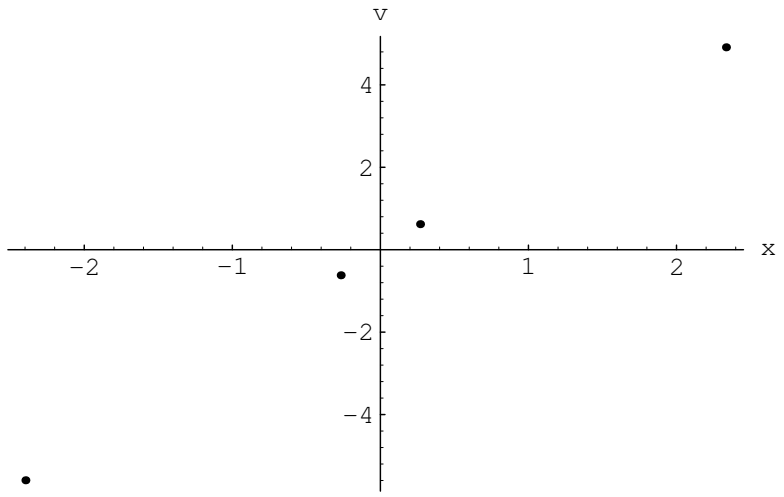


Figure 41: Representation constructed from a good seed representation.

failure points of the task. Figure 41 shows the representation. The learning process which produced it required 207 trials, with the last trial continuing for over 100 million steps.

Control representations

The results compare the performance of the 24-category generated representation with the performance of four other representations: the diagonal-split representation described above, a 10×10 grid partitioning, a representation inspired by Andrew Moore's (1991) Variable Resolution Dynamic Programming, and a representation designed to maintain controllability (Yendo Hu, 1996).

Variable Resolution Dynamic Programming (VRDP) produces a partitioning of the

state-space with the highest resolution at states visited during experimental trials. Away from these experimental trajectories, resolution falls off gradually according to a constraint on neighboring regions. In Moore's work, the experimental trials were "mental practice sessions" conducted according to an internal model being learned by the agent. For the studies reported here, the representation was constructed from two trials using the puck task environment: an initial trial in which the agent always pushed to the right, and a successful trial in which the agent succeeded in keeping the puck in the center of the hill for over 100,000 steps. (The successful trial was taken from a system with the uniform 10×10 partitioning of the space). Because VRDP initializes the representation to a single box, the initial trial consisted of selecting the same action repeatedly (since the policy for all states is the policy of that single box). When the representation was fine enough to allow good performance, mental practice sessions would focus on the states seen in the successful trial. Therefore, VRDP would be likely to visit the same points in mental practice sessions which were visited in the two experimental trials—and most likely, additional points as the representation was being learned and performance was still improving. Therefore, this representation is probably an idealized version of the application of VRDP to the puck task. As in (Moore, 1991), the highest resolution of each state-space coordinate was found by performing six binary splits of that coordinate. Taking the state-space dimensions to be $[-2.4, 2.4] \times [-5.5, 5.5]$, this resulted in the smallest distinctions being $\Delta x = 4.8/64 = 0.075$ and $\Delta v = 11.0/64 = 0.171875$. Figure 42 illustrates the resulting representation.

Unfortunately, this representation performed poorly, attaining a maximum averaged performance score of 2215 steps. (Both the diagonal-split representation and the generated representation achieved averaged scores of 5,000,000 steps). One reason for

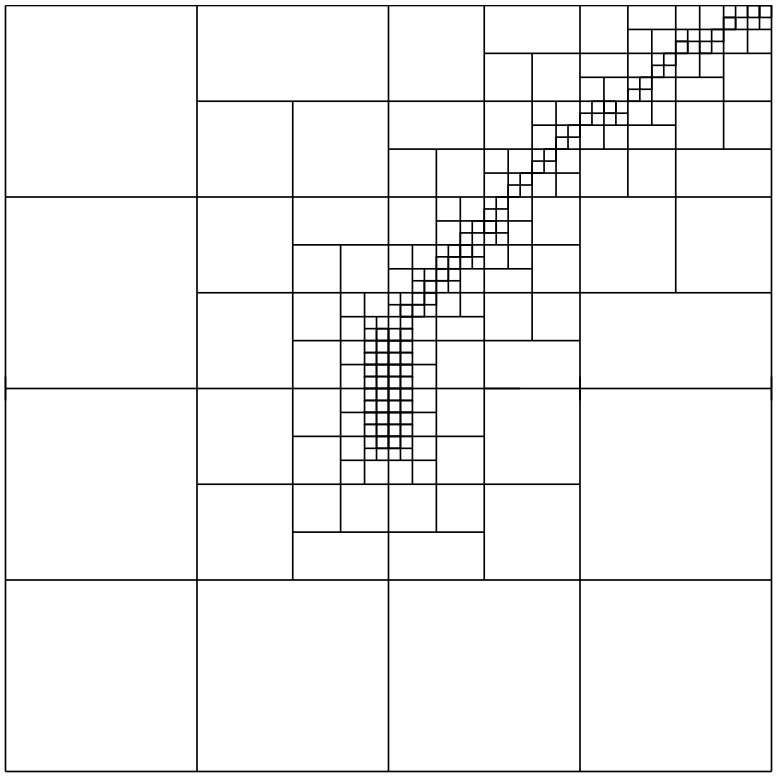


Figure 42: A representation inspired by Variable Resolution Dynamic Programming.

this poor performance may be that the partitioning is very fine along the path from the origin to the failure point of the first trial. As a result, reinforcement from a failure must pass through a very long series of intermediate boxes before it reaches the critical states where the agent can actually control the puck. To test this explanation, I made a second VRDP-inspired representation, shown in Figure 43. Although this representation does not entirely observe the constraint on neighboring regions, it removes most of the boxes resulting from the initial failed trial. Since this representation performed much better than the original, it replaces the original VRDP representation in the comparison plots which follow.

The other representation, shown in Figure 44, was produced by Yendo Hu (1996). This representation quantizes the space according to the puck's sensitivity to control

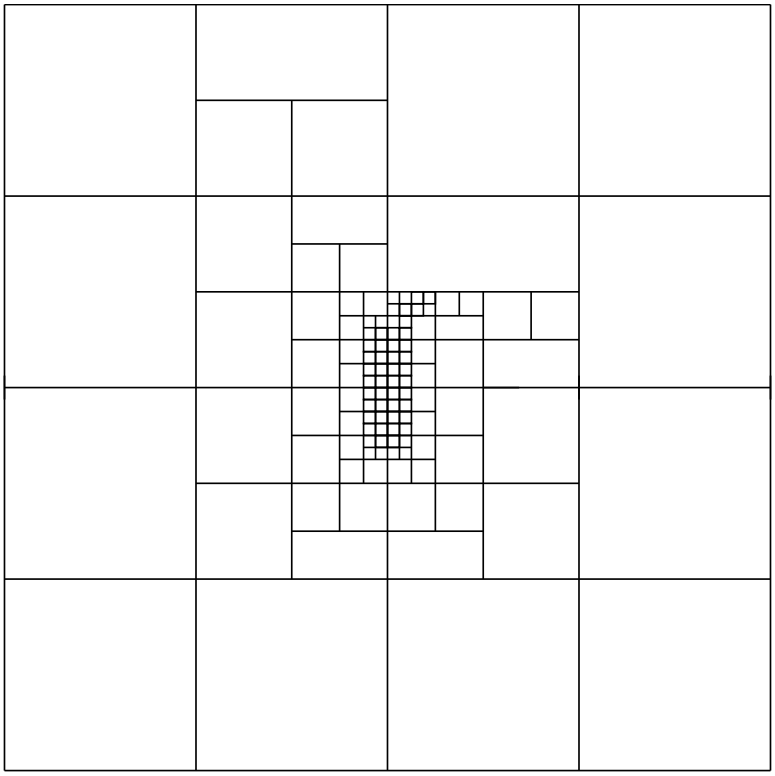


Figure 43: Enhanced VRDP representation.

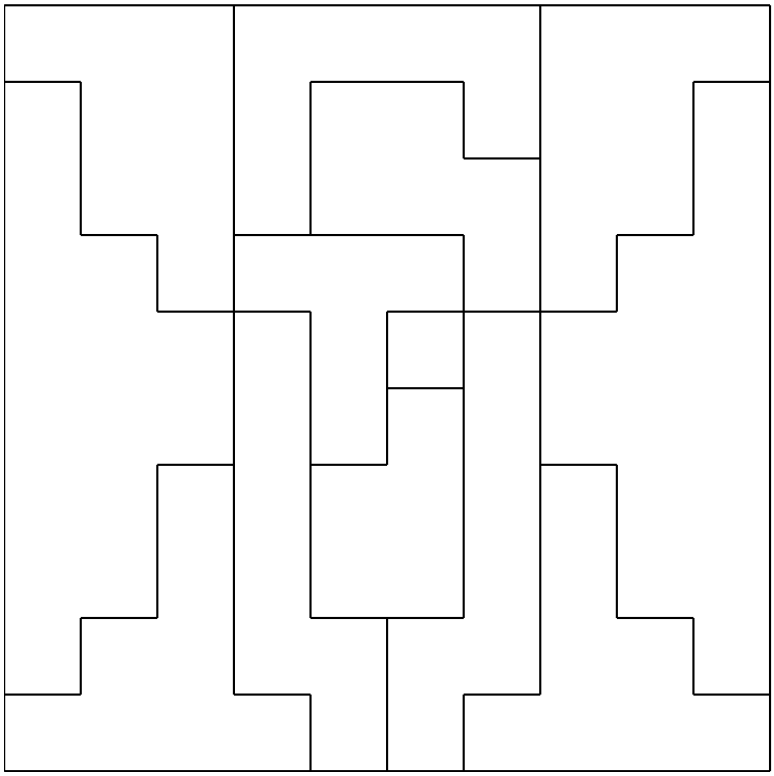


Figure 44: Representation designed to limit the loss of controllability (from Yendo Hu, 1996).

actions. To do this, Hu defined a measure of the loss of controllability: the amount of divergence between trajectories resulting from alternative actions. The partitioning was designed to keep this loss of controllability below a pre-set tolerance. This representation was constructed under the assumption that trials always start at (0, 0). Since the test system starts trials from randomly-chosen starting points, Hu's representation may be at a disadvantage here. This representation was part of an Adaptive Heuristic Critic system (Barto, Sutton, and Anderson, 1983) which learned to balance the puck for over 10,000 steps, after an average of 13 trials and 2000 training steps.

Learning curves

Figure 45 plots the performance for the original VRDP representation (top curve) and Yendo Hu's controllability quantization (bottom curve). Note that the performance scores are all under 2500. Figure 46 shows the averaged curves for the remaining representations. From the top, these are the diagonal-split representation, the representation generated by the learning system, the uniform 10×10 grid, and the enhanced VRDP representation.

The poor performance of the original VRDP representation illustrates an important point: visited states are not necessarily important states. In this task, the important areas of the space are those where the agent's decision makes a critical difference in performing the task.

The cognitive economy approach resulted in a system which was able to automatically construct a good representation from scratch. The representation it constructed had a small number of categories (24), and proved effective in the task. When given an effective seed representation, the system made minimal additions, indicating an ability to discern relevant distinctions.

6.5 Case Study: Pole Balancing Task

The pole balancing task involves a wheeled cart on a track, with a pole hinged to the top of the cart, as shown in Figure 47. At each step, the controller must decide whether the cart should apply a fixed force to the left or to the right, in order to keep the pole balanced vertically.

The equations of motion are as follows, with angles in radians. These may also be

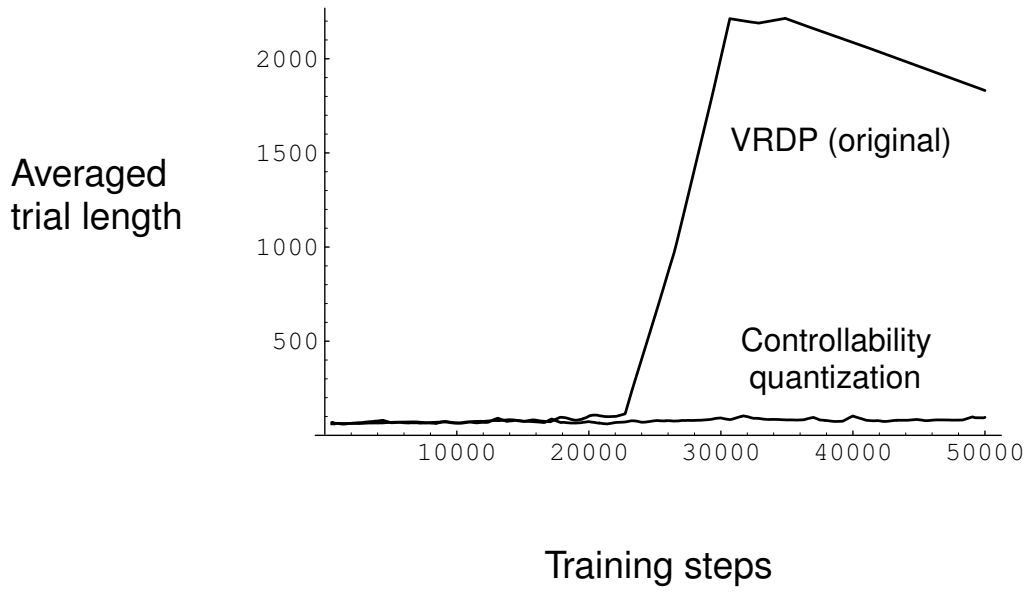


Figure 45: Averaged performance curves for the original VRDP representation and Yendo Hu's controllability quantization.

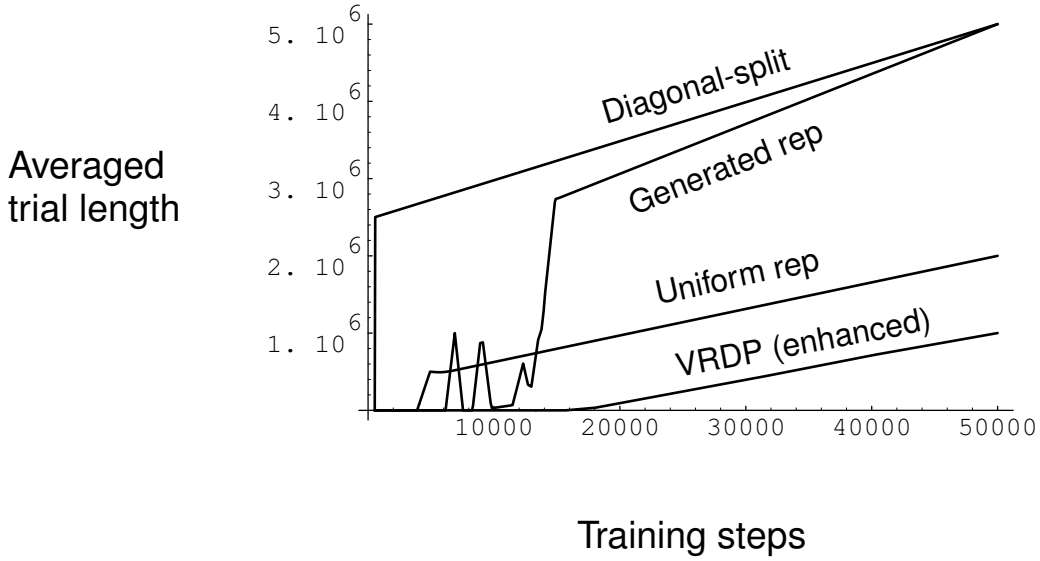
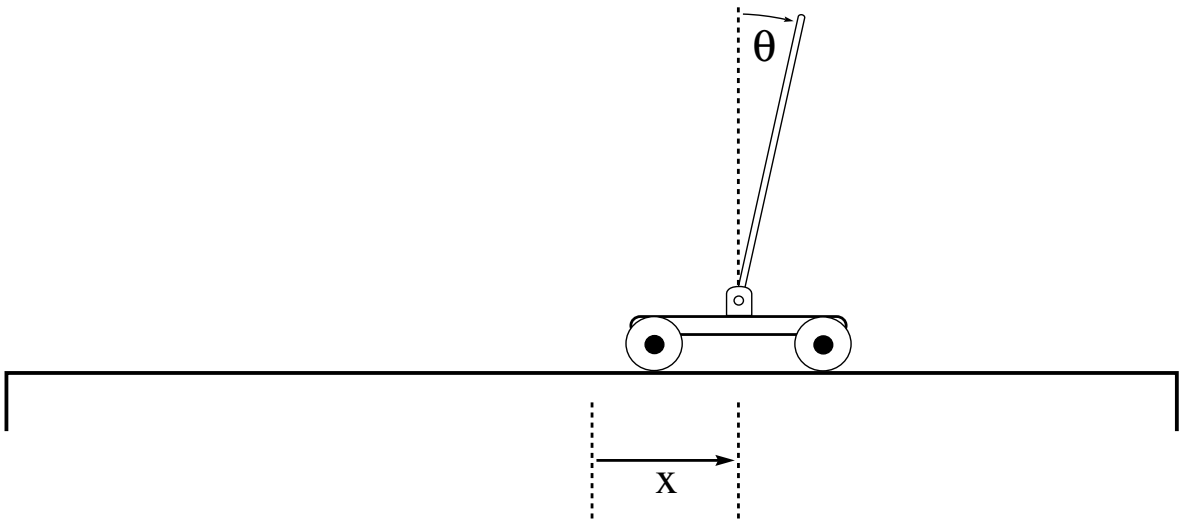


Figure 46: Averaged performance curves for the four best representations.



State	x, position of cart (meters from center of track)
	$\dot{x}$, velocity of cart (meters / second)
	θ, angle of pole from vertical
	$\dot{\theta}$, angular velocity of pole
Control	f, force on cart (Newtons)
Constraints	$-2.4 < x < 2.4$
	$-12^\circ < \theta < 12^\circ$
	$f = \pm 10.0$
Parameters	$g = 9.8$ m / s^2, gravitational acceleration
	$m = 1.1$ kg, combined mass of cart and pole
	$m_p = 0.1$ kg, mass of pole
	$l = 0.5$ m, distance from pivot to pole's center of mass
	$\Delta = 0.02$ s, sampling interval

Figure 47: The cart-pole apparatus. The task is to balance the pole by pushing the cart to either the left or the right in each control interval.

found in Anderson and Miller's (1990) collection of "Challenging Control Problems."

$$
\begin{aligned}
\theta(t+1) &= \theta(t) + \Delta\dot{\theta}(t) \\
\dot{\theta}(t+1) &= \dot{\theta}(t) + \Delta\ddot{\theta}(t) \\
\ddot{\theta}(t) &= \frac{mg\sin\theta(t) - \cos\theta(t)\left(f(t) + m_p\, l\,(\dot{\theta}(t))^2\,\sin\theta(t)\right)}{(4/3)m\,l\, - m_p\, l\,\cos^2\theta(t)} \\
x(t+1) &= x(t) + \Delta\dot{x}(t) \\
\dot{x}(t+1) &= \dot{x}(t) + \Delta\frac{f(t) + m_p\, l\,\left((\dot{\theta}(t))^2\,\sin\theta(t) - \ddot{\theta}(t)\cos\theta(t)\right)}{m}
\end{aligned}
$$

Note that positive x values are toward the right, positive θ indicates that the pole is falling toward the right, and positive f pushes the cart toward the right. Failure occurs when the pole falls too far to either the left or the right ($|\theta| \geq 12$ degrees), or when the cart falls off the track ($|x| \geq 2.4$). Upon failure, the current trial ends and the environment sends the system a reinforcement signal $r(t) = -1$. Otherwise, the system sees no reinforcement from the environment.

6.5.1 Analysis

In the puck-on-a-hill task the obstacle to balancing the puck is the force of gravity, which increases as the puck falls farther from the center. Beyond a fairly narrow central region, gravity overpowers the puck's thrusters, making balancing impossible. The situation is different for the pole balancing task. Here the cart's thrust is always sufficient to dominate the longitudinal force given by the fall of the pole. We can see this by plotting the values of $\ddot{\theta}$ for θ and $\dot{\theta}$ within the ranges seen in the task. Figure 48 plots the values of $\ddot{\theta}$ for $\theta \in [-12°, 12°]$, $\dot{\theta} \in [-190°, 190°]$, and positive thrust f. For $f = 10, \ddot{\theta} \in [-17.38, -11.16]$. For $f = -10, \ddot{\theta} \in [11.16, 17.38]$. Notice that the force on

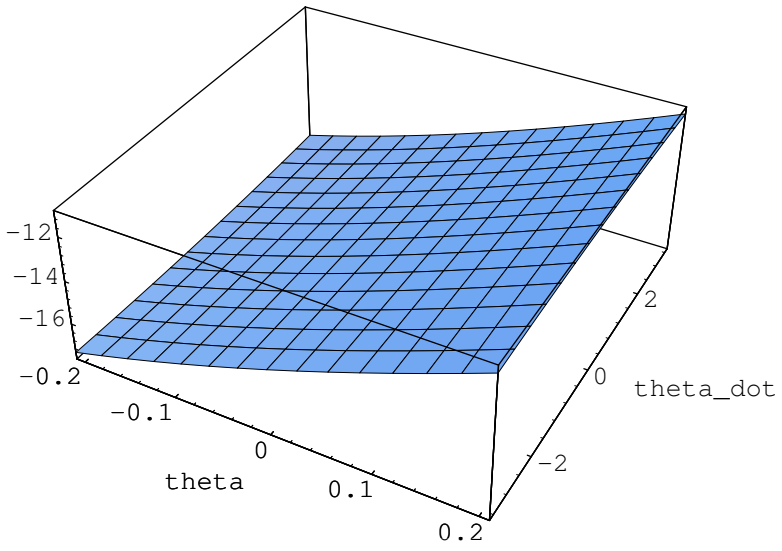


Figure 48: Angular acceleration for the pole, $f = 10.0$.

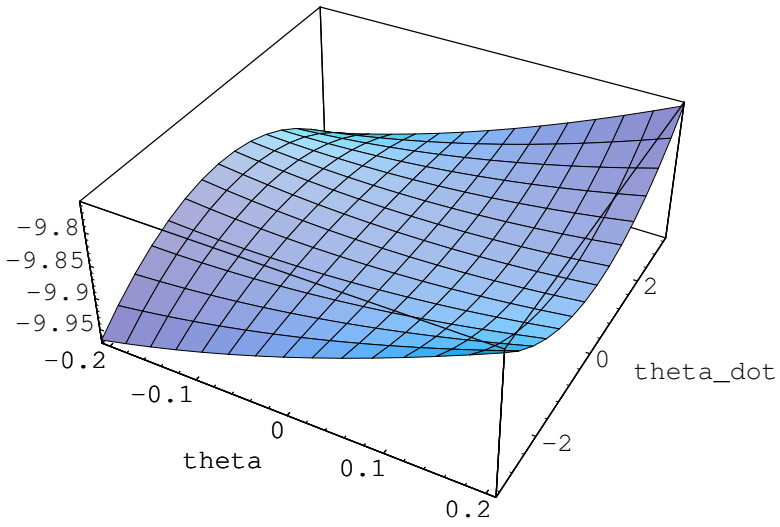


Figure 49: Acceleration of the cart, for $f = -10.0$ and $\ddot{\theta} = 17.38$.

the pole does not depend on the position or speed of the cart.

Likewise, the force on the cart itself is mainly determined by the cart's own thrusters. Figure 49 shows the lowest values of $\ddot{x}$, which are obtained when $f = -10$ and $\ddot{\theta}$ assumes its highest value of 17.38. Figure 50 has the same shape, and shows the highest values of $\ddot{x}$, achieved for $f = 10$ and $\ddot{\theta} = -17.38$. Note that $\ddot{\theta}$ and its contribution to $\ddot{x}$ have opposite signs, because when the pole falls to one side it accelerates the cart in the opposite direction. These acceleration plots show that pushing the cart to the right always accelerates the cart to the right, and the pole to the left. Pushing to the left always accelerates the cart to the left and the pole to the right. Unlike the puck task, the agent can always change the direction of either the cart's movement or the pole's falling. The task is difficult because combining the two sub-tasks—balancing the pole

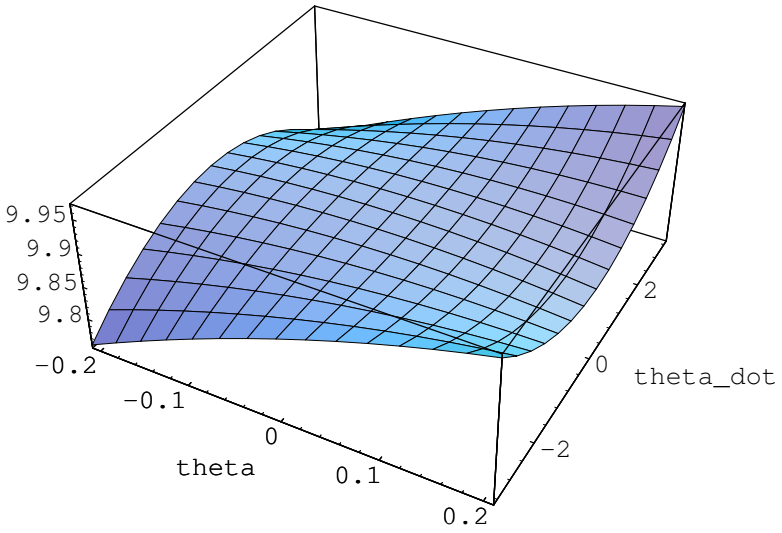


Figure 50: Acceleration of the cart, for $f = 10.0$ and $\ddot{\theta} = -17.38$.

and centering the cart—sometimes requires the agent to satisfy conflicting goals.

If balancing the pole and centering the cart require opposite actions, the agent may be unable to bring the system back to equilibrium. In addition, the interaction of cart and pole can result in the need to take what seems to be the "wrong" action for either one taken separately. For an example of this interaction, suppose that the system begins at state $(x, \dot{x}, \theta, \dot{\theta}) = (0, 0, 0, 0)$ and executes a series of n right pushes, the pole will begin falling to the left. To re-balance the pole requires more than n left pushes because of the pole's momentum, acquired during the fall, and the force of gravity which must be overcome during the recovery. Balance is restored, but with the side-effect of accelerating the cart to the left. This strategy allows the agent to center the cart, by first pushing it in the "wrong" direction. The need for such counter-intuitive strategies is strongest at the extreme values of the state-space, where the pole is falling and the cart is running out of track. Here all the wrong decisions made earlier in the trial finally have their payoff. Since the actions which set up this chain of events may have been at the beginning of a very long trial, finding and correcting the faulty actions can be very difficult.

The task is easiest to solve when started from states far from the edges of the track. Geva and Sitte (1993) have shown that in states near the equilibrium point, a single linear decision rule suffices to keep the system balanced: push right if $0.05\,x + 0.1\,\dot{x} + 0.94\,\theta + 0.33\,\ddot{\theta} > 0.0$. For the case when the cart is close to the center of the track and moving slowly, this represents the half-plane above the falling line $\ddot{\theta} = -2.85\,\theta$ in $(\theta, \ddot{\theta})$-space. The strategy is the obvious one: push to the right when the pole has fallen or is falling to the right; otherwise, push left.

6.5.2 Results

The pole task appears to be much more difficult than the puck task, and the ideal state generalization is not obvious. The objectives of this study were to verify that feature extraction based on cognitive economy could solve more complex tasks than the puck task. The specific goals were to generate the representation automatically, achieving a high standard of performance and a small number of feature detectors.

The learning system generated its representation from scratch, producing a set of 26 feature detectors which allowed it to balance the pole for over 100 million steps, after 6000 trials. The majority of the detector regions included a single nearest-neighbor prototype, although some regions held as many as 13 prototype points which had been merged.

This representation was then compared with the 162-box representation used in Barto, Sutton, and Anderson (1983) and other studies. The 162-box representation was not learned on-line, but was hand-designed for studies of the effectiveness of various algorithms for learning action values. It appears to be the result of thoughtful consideration of the task dynamics; for example, it devotes particular attention to small changes in the pole angle about the equilibrium point. Both the 162-box representation and the generated representation were inserted into a test system which then learned the task in a series of experiments. Each experiment consisted of 20,000 trials, with the length of the best trial recorded as a performance score. Ten such experiments were run for each of the two representations; the average of the ten performance scores was taken as the resulting performance measurement for that representation. The averaged score for the generated representation was 3,054,117 steps, and the score for the 162-box representation was 4,224,068 steps. The test system employed the enhanced Q-learning

algorithm described earlier, with trials beginning from randomly-chosen starting points. Since $\Delta t = 0.02$ seconds, every 100,000 steps represents about 33 minutes of real-world balancing time.

These numbers compare favorably with typical results from the literature: Anderson (1986) achieved average trial lengths of about 28,000 steps with a two-layer back-propagation neural network having 70 weights; Barto, Sutton, and Anderson (1983) achieved average trial lengths of nearly 80,000 steps with the Adaptive Heuristic Critic algorithm and the 162-box representation; Hu and Fellman (1996) achieved close to 200,000 step average trial lengths with a representation having 105 boxes; Munos and Moore (1999) solved a related cart-pole task (minimum time to a specified goal region), but with 40,000 to 80,000 boxes in the representation.

The more significant finding was that the generated representation supported performance at a level comparable to a representation which had been hand-designed for the task.

6.6 Discussion

The purpose of these experiments was to verify the soundness of the ideas developed in earlier chapters. The results indicate that the ideas are successful and useful. The system presented in this chapter was able to learn better representations for the puck task than those used by others, and it learned a representation for the pole task which was comparable to one which had been hand-designed for the task. Furthermore, the generated representations were smaller, and they were generated from scratch. This success supports the conclusion that cognitive economy can be successfully applied to reinforcement learning, leading to the automatic generation of effective representations.

In particular, this chapter demonstrated the usefulness of selecting relevant training examples by the criterion for surprising states (representational adequacy), as well as splitting and merging regions according to state compatibility. These criteria allow the system to focus on the distinctions that are relevant to the task at hand. Developing representations which focus on relevant distinctions is one of the abilities needed by reinforcement learning systems which learn complex tasks in unknown domains. This chapter is a first step in the application of these ideas.

6.7 Future Work

The system studied in this chapter serves as a starting point for further research. Besides further tuning and balancing of the components of the system, several alternative implementations show promise as ways of extending these ideas to a wider range of tasks: using the test for surprising states as a trigger for state splitting, and the proposed idea for constructing an internal model which the agent can query in place of the active investigations. In addition, the active investigation algorithm should be studied as a possible way of learning certain tasks more efficiently than traditional methods.

The feature extraction ideas which were tested here may shed light on new ways of constructing representations in systems which are entirely policy-based, or which lift the restriction that the representation be a partition. These extensions require additional development of the underlying theory, but will allow application to many important tasks.

Chapter 7

Conclusion

This dissertation examined the role of representation in reinforcement learning: it demonstrated how the principle of cognitive economy may be formulated in terms of action values, and how the resulting criteria can be implemented successfully to achieve on-line construction of new representations. Although other formulations are possible, the analysis presented here showed that these criteria are not ad-hoc, but establish a well-defined standard of learnability. Similarly, although the case studies are only a starting point for the implementation of these criteria, they show that cognitive economy may be successfully applied to reinforcement learning problems.

Representation is fundamental to cognition and remains an important open problem. Agents learning large and complex tasks require efficient representations, characterized by high cognitive economy: such representations allow the agent to make necessary distinctions, but avoid presenting the agent with distinctions which do not matter in its task. Chapter 3 formalized the principle of cognitive economy, and developed a framework for defining important features and necessary distinctions. That framework ties the adequacy of a representation to the agent's ability to choose actions which result in low incremental regret—a formalization of the requirement of making "sound decisions" in the task. Chapter 4 continued the analysis by defining the true action values in a task—the *generalized action values*—and showed how these values depend

on the representation. Building on this result, Chapter 5 proved that partition representations which separate incompatible states meet the ϵ-adequacy standard and thus allow the agent to learn to make sound decisions in the task. The case studies of Chapter 6 present an algorithm for on-line feature extraction which successfully applied these criteria to automate the construction of effective representations for a difficult reinforcement learning problem.

7.1 Contributions

This dissertation introduced a new conceptual framework for exploring the role of representation. The key concepts include the following: generalized action values, action preference sets, feature importance, pure and mixed sets of states, sound decisions, policy and value distinctions, incremental regret, state compatibility, and representational adequacy. The key ideas are these:

- The representation changes the nature of the task by the way that it generalizes action values over states. Without any state generalization, tasks in continuous worlds are simply infeasible. State generalization reduces the amount of experience needed to learn the task by sharing the learning among groups of states. In this way, state generalization controls which states are treated as "the same kind of thing." We want to group together states which are "the same" with respect to the agent's task, but still allow the agent to distinguish states which require different policies or which lead to significantly different outcomes.

- Some distinctions between states are irrelevant to the agent's decisions, even

though they would be necessary to accurately predict all the action values. Cognitive economy leads us to filter out these irrelevant distinctions, unlike previous approaches.

- Feature importance is tied to the ability of a feature to make a state-space distinction which matters to the agent in its task. We can define importance in terms of generalized action values.

- A representation which generalizes over incompatible states is inadequate for the task. We can define compatibility criteria so that the loss incurred by any failure to distinguish states is always less than a given tolerance—our standard for learnability.

- Function approximation for reinforcement learning needs to take state compatibility into account, including differences in preferred policy between states. This is necessary in order to respond to action value errors which are important, while ignoring benign errors which do not matter in the task.

Learning the representation along with the task is especially challenging. It requires effective criteria for selecting relevant states for consideration, and for determining when states may be considered similar in the context of the task. The dissertation presented an algorithm built on the new framework, and demonstrated that it is able to learn effective representations at the same time it is learning how to behave in the world. This algorithm is a starting point for future work which applies cognitive economy to reinforcement learning.

7.2 Future work

This work suggests several areas for the continued development of our understanding of representation and its role in on-line learning. In this dissertation, I have often chosen to define the general principles in straight-forward ways which permitted the clearest exposition of the concepts and allowed me to prove some important properties for the type of system I wanted to build. The most important topic for future work might be the extension of the criteria for representational adequacy and state compatibility to the case where the representation is not a partition. In particular, the ideas might be extended to the case where different feature detectors apply to different actions in the task. This extension would allow application of the ideas to more tasks.

Another interesting area for extension is Chapter 5's proof that state compatibility leads to representational adequacy. This proof gave sufficient conditions for partition representations; it would be very interesting to see it extended to coarse-coded representations, and it would be very interesting to see a result for necessary and sufficient conditions.

The idea of feature importance appears to be related to beneficial state generalization, over pure sets. This should be explored in more depth, along with the development of more sophisticated definitions of feature importance.

The implementation of Chapter 6 can be the basis for future applications and experiments. First, the active learning algorithm is interesting in its own right, as a system which may solve some tasks more efficiently than traditional methods; it selects as training data states which are on the frontier of that portion of the task which has already been learned.

Non-active implementations could extend the applicability of the cognitive economy

approach to a wider variety of tasks. Chapter 6 presented two promising non-active schemes: simply adding a new prototype state when the current experience is sufficiently surprising (and merging states which later prove compatible), and saving all the agent's experiences into categories for the current regions, and using them to construct an internal model for queries about state compatibility.

It will help to apply these ideas to additional tasks of greater complexity. Producing working systems requires the achievement of a balance and synergy between their parts. In this case, there needs to be a balance between learning the action values and learning the representation, between the top-level Q-learning which smooths out the values and the active investigations, between splitting regions and consolidating them, and between increasing the accuracy of the regions' boundaries and producing small, economical representations. These are practical issues which are best studied experimentally.

This work could also lead to research in the field of cognitive psychology, since it makes claims regarding the fitness of different kinds of representations for particular tasks. Because experts appear to develop representations which show high cognitive economy, it would be interesting to apply the ideas of this dissertation to evaluate expert representations, the effect of representation on human performance, and the potential for finding better representations for the tasks people face. The study of reinforcement learning systems has allowed me to state these ideas objectively and to analyze them mathematically. But human cognition appears to work according to similar principles of cognitive economy, since our pre-conscious awareness appears to be filtered by categorization (Harnad, 1987; Berlin and Kay, 1969), and our conscious decisions appear to depend on simplifying generalizations (Simon, 1957). If we would draw conclusions about intelligent action *in general,* we need experiments and analysis

with both natural and artificial subjects.

Afterword

> Our acceptance of an ontology is, I think, similar in principle to our acceptance of a scientific theory, say a system of physics; we adopt, at least insofar as we are reasonable, the simplest conceptual scheme into which the disordered fragments of raw experience can be fitted and arranged.
>
> —*Willard V. O. Quine*

> Thus the order and regularity in the appearances, which we entitle *nature,* we ourselves introduce.
>
> —*Immanuel Kant*

Representation and feature extraction are fundamental to intelligence. The ideas we have been considering in a very tightly-constrained model underlie most of human cognition. Therefore, it is important to consider the place of this discussion in the big picture. Early in life, we learn to categorize our perceptions in order to interact successfully with our world. For example, we become sensitized to the speech sounds which occur in our native tongue, and to the visual stimuli of our parents' faces. As we grow, we learn "what to look for" in order to perform various tasks, learning which data are relevant to our needs. We learn to categorize the world according to our goals, and this categorization reduces the complexity of our world to manageable levels, enabling us to act intelligently. But since this categorization filters our view of the world, it removes us from reality itself.

In Zen and the Art of Motorcycle Maintenance, Robert Pirsig describes this categorization as the action of a knife which we use to slice reality:

> The application of this knife, the division of the world into parts and the building of this structure, is something everybody does. All the time we are aware of millions of things around us—these changing shapes, these burning hills, the sound of the engine, the feel of the throttle, each rock and weed and fence post and piece of debris beside the road—aware of these things but not really conscious of them unless there is something unusual or unless they reflect something we are predisposed to see. We could not possibly be conscious of these things and remember all of them because our mind would be so full of useless details we would be unable to think. From all this awareness we must select, and what we select and call consciousness is never the same as the awareness because the process of selection mutates it. We take a handful of sand from the endless landscape of awareness around us and call that handful of sand the world.
>
> Once we have the handful of sand, the world of which we are conscious, a process of discrimination goes to work on it. This is the knife. We divide the sand into parts. This and that. Here and there. Black and white. Now and then. The discrimination is the division of the conscious universe into parts.

He goes on to describe how each "grain of sand" is unique, and we can sort them into piles on the basis of all kinds of different properties. And the properties we choose for this sorting depend on the analogs we have acquired from our experience, and from the collective experience passed on to us through human society.

The Zen approach to categorization is to refuse to separate the sand into piles, since

doing so inevitably removes us from Reality. For then we no longer see the world—only the categories. As a result, the world we interact with is an artificial construct, designed according to our needs and concerns. But, to a large extent, we cannot help it—we cannot survive without doing so. Most of our education and growth concerns our ability to categorize, and to discern the connections between categories. If these categories are poorly chosen, we make mistakes, or else we work much harder than necessary to make correct decisions. Therefore, we must be prepared to throw away our categories and to re-categorize experience when our representation proves to be inadequate.

If I claim that reward-maximization and cognitive economy govern human decision-making, I must be quick to point out that these are only part of the picture. My model is based on the rewards experienced by an agent as the result of its actions, but says nothing at all about the source of those rewards, or how they are determined. The rewards appear to encapsulate everything that is involved in being human. Having bodies, we experience rewards from sensations such as pleasure and pain, and find that our actions are constrained by the limitations of our bodies. Living in families and societies, we experience other types of rewards based on social interaction. Religious faith allows for a transcendent source of reward and value. Clearly, the desirability of various experiences and states depends on a complex web of environmental, social, cultural and spiritual stimuli and rewards. All of this is outside my model of information processing—and possibly beyond formal description. This dissertation has studied some of the central issues of cognition, but only in a very limited model. It is a beginning, but the kind of categorization which we take for granted as human beings going about our normal lives will probably always be seen as art.

Bibliography

[1] Douglas Adams. *Mostly Harmless*. Harmony Books, New York, NY, 1986.

[2] James S. Albus. *Brains, Behavior, and Robotics*. Byte Books, Peterborough, New Hampshire, 1981.

[3] Charles W. Anderson. *Learning and Problem Solving With Multilayer Connectionist Systems*. PhD thesis, University of Massachusetts, Amherst, Massachusetts, 1986.

[4] Charles W. Anderson and W. Thomas Miller, III. A challenging set of control problems. In W. Thomas Miller, III, Richard S. Sutton, and Paul J. Werbos, editors, *Neural Networks for Control*, pages 475–510. MIT Press, Cambridge, Massachusetts, 1990.

[5] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13:834–846, 1983.

[6] Brent Berlin and Paul Kay. *Basic Color Terms*. University of California Press, Berkeley, CA, 1969.

[7] Donald A. Berry and Bert Fristedt. *Bandit Problems: Sequential Allocation of Experiments*. Chapman and Hall, London, UK, 1985.

[8] Marc H. Bornstein. Perceptual categories in vision and audition. In Stevan Harnad, editor, *Categorical Perception*, chapter 9. Cambridge University Press, Cambridge, 1987.

[9] Gail A. Carpenter and Stephen Grossberg. The ART of adaptive pattern recognition by a self-organizing neural network. *Computer*, 21(3):77–88, March 1988.

[10] David Chapman and Leslie Pack Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In John Mylopoulos and Ray Reiter, editors, *Proceedings of the Twelfth International Joint*

Conference on Artificial Intelligence (IJCAI-91), pages 726–731, San Mateo, Ca., 1991. Morgan Kaufmann.

[11] A. M. Collins and M. R. Quillian. Retrieval time from semantic memory. *Journal of Verbal Learning and Verbal Behavior*, 8:240–247, 1969.

[12] Carol Conrad. Cognitive economy in semantic memory. *Journal of Experimental Psychology*, 92(2):149–154, 1972.

[13] R. H. Crites and A. G. Barto. Improving elevator performance using reinforcement learning. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference*, pages 1017–1023, Cambridge, MA, 1996. MIT Press.

[14] Adriaan D. de Groot. *Thought and Choice in Chess*. Mouton & Co, The Hague, 1965.

[15] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, New York, 1973.

[16] David J. Finton and Yu Hen Hu. An application of importance-based feature extraction in reinforcement learning. In J. Vlontzos, J-N. Hwang, and E. Wilson, editors, *Neural Networks for Signal Processing*, volume IV, pages 52–60. IEEE Press, Piscataway, NJ, September 1994.

[17] David J. Finton and Yu Hen Hu. Importance-based feature extraction for reinforcement learning. In *Computational Learning Theory and Natural Learning Systems*, volume III: Selecting Good Models, chapter 5, pages 77–94. MIT Press, 1995.

[18] Shlomo Geva and Joaquin Sitte. A cartpole experiment benchmark for trainable controllers. *IEEE Control Systems*, 13(5):40–51, October 1993.

[19] Stevan Harnad. Introduction: Psychophysical and cognitive aspects of categorical perception: A critical overview. In Stevan Harnad, editor, *Categorical Perception*. Cambridge University Press, Cambridge, 1987.

[20] Geoffrey E. Hinton. Distributed representations. Technical Report CMU-CS-84-157, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1984.

[21] Douglas R. Hofstadter. *Gödel, Escher, Bach : an Eternal Golden Braid*. Basic Books, New York, NY, 1979.

[22] R.M. Holdaway. Enhancing supervised learning algorithms via self-organization. In *Proceedings of the International Joint Conference on Neural Networks*, pages 523–529, 1989.

[23] J. H. Holland. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 2, chapter 20. Morgan Kaufmann Publishers, San Mateo, Ca, 1986.

[24] Yendo Hu. *Reinforcement Learning for Dynamic Robotic Systems*. PhD thesis, University of California, San Diego, CA, 1996.

[25] Yendo Hu and Ronald D. Fellman. An efficient adaptive input quantizer for resetable dynamic robotic systems. In *Proceedings of the IEEE International Conference on Neural Networks, Washington, DC (ICNN-96)*. IEEE, 1996.

[26] Leslie Pack Kaelbling. Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 167–173, San Mateo, CA, 1993. Morgan Kaufmann.

[27] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–277, 1996.

[28] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[29] Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

[30] Aristidis Likas. Reinforcement learning using the stochastic fuzzy min-max neural network. *Neural Processing Letters*, 13:213–220, 2001.

[31] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321, 1992.

[32] Richard Maclin and Jude W. Shavlik. Creating advice-taking reinforcement learners. *Machine Learning*, 22:251–281, 1996.

[33] Andrew Kachites McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, Rochester, NY, 1995.

[34] D. Michie and R. A. Chambers. Boxes: An experiment in adaptive control. In E. Dale and D. Michie, editors, *Machine Intelligence 2*. Oliver and Boyd, Edinburgh, 1968.

[35] Tom M. Mitchell. *Machine Learning*. McGraw Hill, New York, 1997.

[36] J. Moody and C. J. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294, 1989.

[37] Andrew W. Moore. Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued spaces. In *Proceedings of the Eighth International Machine Learning Workshop*, 1991.

[38] Andrew W. Moore and Christopher G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13:103–130, 1993.

[39] Andrew W. Moore and Christopher G. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21, 1995.

[40] Donald A. Norman. *Things That Make Us Smart*. Addison-Wesley, Reading, Massachusetts, 1993.

[41] Robert M. Pirsig. *Zen and the Art of Motorcycle Maintenance*. Bantam Books, New York, 1974.

[42] T. Poggio and F. Girosi. Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, 247:978–982, 1990.

[43] Rémi Munos and Andrew Moore. Variable resolution discretization for high-accuracy solutions of optimal control problems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99)*, 1999.

[44] Nicholas Rescher. *Cognitive Economy: The Economic Dimension of the Theory of Knowledge*. University of Pittsburgh Press, Pittsburgh, PA, 1989.

[45] Eleanor Rosch. Principles of categorization. In Eleanor Rosch and Barbara B. Lloyd, editors, *Cognition and Categorization*, chapter 2, pages 27–48. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1978.

[46] Frank Rosenblatt. *Principles of Neurodynamics*. Spartan, New York, 1962.

[47] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, 3:211–229, 1959.

[48] Herbert A. Simon. *Models of Man: Social and Rational*. John Wiley & Sons, Inc, New York, 1957.

[49] Satinder P. Singh, Tommi Jaakkola, and Michael I. Jordan. Reinforcement learning with soft state aggregation. In G. Tesauro, D. S. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems: Proceedings of the 1994 Conference*, pages 361–368. MIT Press, Cambridge, MA, 1995.

[50] Charles T. Snowdon. A naturalistic view of categorical perception. In Stevan Harnad, editor, *Categorical Perception*, chapter 11. Cambridge University Press, Cambridge, 1987.

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

[52] R. S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[53] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning*. MIT Press, Cambridge, MA, 1998.

[54] Gerald Tesauro. Temporal difference learning and TD-gammon. *Communications of the ACM*, 38(3):58–67, March 1995.

[55] E. L. Thorndike. *Animal Intelligence*. Hafner, Darien, CT, 1911.

[56] L. Z. Wang and J. V. Hanson. Competitive learning and winning-weighted competition for optimal vector quantizer design. In C. A. Kamm, G. M. Kuhn, B. Yoon,

R Chellappa, and S. Y Kung, editors, *Neural Networks for Signal Processing III: Proceedings of the 1993 IEEE Workshop*, pages 50–59. IEEE Press, 1993.

[57] C. J. C. H. Watkins. *Learning From Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, England, 1989.

[58] C.J.C.H. Watkins and P. Dayan. Technical note: Q-learning. *Machine Learning*, 8(3/4):279–292, 1992.

[59] Joseph Weizenbaum. *Computer Power and Human Reason*. W. H. Freeman and Company, 1976.

[60] Janet F. Werker and Richard C. Tees. Cross-language speech perception: Evidence for perceptual reorganization during the first year of life. *Infant Behavior and Development*, 7:49–63, 1984.

[61] W. Zhang and T. G. Dietterich. High-performance job-shop scheduling with a time-delay TD(λ) network. In *Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference*, pages 1024–1030, Cambridge, MA, 1996. MIT Press.