Dynamic Generalisation of Continuous Action Spaces in Reinforcement Learning: A Neurally Inspired Approach

Andrew James Smith

Doctor of Philosophy

Institute for Adaptive and Neural Computation
Division of Informatics
University of Edinburgh

October 2001

To Mum and Dad

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise, and that this work has not been submitted for any other degree or professional qualification.

(*Andrew James Smith*)

Abstract

This thesis is about the dynamic generalisation of continuous action spaces in reinforcement learning problems.

The standard Reinforcement Learning (RL) account provides a principled and comprehensive means of optimising a scalar reward signal in a Markov Decision Process. However, the theory itself does not directly address the imperative issue of generalisation which naturally arises as a consequence of large or continuous state and action spaces. A current thrust of research is aimed at fusing the generalisation capabilities of supervised (and unsupervised) learning techniques with the RL theory. An example par excellence is Tesauro's *TD-Gammon*.

Although much effort has gone into researching ways to represent and generalise over the input space, much less attention has been paid to the action space. This thesis first considers the motivation for learning real-valued actions, and then proposes a set of key properties desirable in any candidate algorithm addressing generalisation of both input and action spaces. These properties include: Provision of adaptive and online generalisation, adherence to the standard theory with a central focus on estimating expected reward, provision for real-valued states and actions, and full support for a real-valued discounted reward signal. Of particular interest are issues pertaining to robustness in non-stationary environments, scalability, and efficiency for real-time learning in applications such as robotics. Since exploring the action space is discovered to be a potentially costly process, the system should also be flexible enough to enable maximum reuse of learned actions.

A new approach is proposed which succeeds for the first time in addressing all of the key issues identified. The algorithm, which is based on the ubiquitous self-organising map, is analysed and compared with other techniques including those based on the backpropagation algorithm. The investigation uncovers some important implications of the differences between these two particular approaches with respect to RL. In particular, the distributed representation of the multi-layer perceptron is judged to be something of a double-edged sword offering more sophisticated and more scalable generalising power, but potentially causing problems in dynamic or non-equiprobable environments, and tasks involving a highly varying input-output mapping.

The thesis concludes that the self-organising map can be used in conjunction with current RL theory to provide real-time dynamic representation and generalisation of continuous action spaces. The proposed model is shown to be reliable in non-stationary, unpredictable and noisy environments and judged to be unique in addressing and satisfying a number of desirable properties identified as important to a large class of RL problems.

Acknowledgements

I would firstly like to thank my supervisors, David Willshaw and John Hallam, for their guidance throughout this project. I am particularly grateful for feedback regarding draft versions of this thesis. My original supervisor, Bruce Graham, also receives my hearty thanks for being interested enough in my ideas to offer me both a position in the department and his services as a supervisor. A particular gratitude I owe Bruce is for his frequent provision of URLs, paper titles and books that he thought I might be interested in, without a number of which I believe this thesis would be of significantly lower quality. Special thanks also go to my M.Sc. supervisor, Mark Ellison, who in my opinion went far beyond his obligations either as course organiser or my personal supervisor, to help identify, encourage and develop my current research interests. This included an introduction to Joanna Bryson who later helped me stay sane while I was waiting to apply for a Ph.D. place, with a large volume of email correspondence on the subject of learning and robots. It is more than possible that at places during this document I have used ideas that we discussed during these periods without explicit acknowledgement.

I would also like to acknowledge technical assistance from a number of fellow members of the recently formed Institute for Adaptive and Neural Computation at Edinburgh university. In particular, Amos Storkey for helping me thrash out some technical analyses, Chris Williams for a number of helpful discussions including one in particular on the GTM algorithm, Matthias Seeger for talking through and providing technical guidance and some useful definitions on the mechanics of projectiles in chapter 7, and also Nick Adams for many technical and non-technical discussions, and instant solutions to most of my computer related problems. Particular thanks are also due to Alexander Holt, LaTeX guru extraordinaire, and to Grandma for buying all those raffle tickets so many years ago and getting me started on my first computer.

This work was carried out courtesy of a research grant from the Engineering and Physical Sciences Research Council. Award number: 98318242.

Contents

List of Figures

CHAPTER 1

Introduction

The ability of a system to adapt to its surroundings and improve its performance through exposure to a problem is ubiquitous in nature. Examples include animal brains, immune systems, the evolution of genetic material, the laws and languages of a society, the strengthening of muscle tissue in response to exercise, and even the way in which a plant grows towards a light. These or related ideas have been modelled extensively by researchers in a variety of areas including machine learning, genetic algorithms, reinforcement learning, neuroscience, psychology, ethology, biology and computer science.

The notion of adaptation is also commonplace in our own engineered systems, where examples include computer programs that learn to play games, robots that learn to produce behaviour, adaptive plant control, and a host of applications that depend on various optimisation, density estimation, regression, and classification techniques.

The field is large and therefore further distinctions are useful. For example, does adaptation occur *in-lifetime* or *outside-lifetime*? Is learning *supervised*, *unsupervised* or *reinforced*? Is the system *natural* or *artificial*? If artificial, is the model *symbolic* or *sub-symbolic*? Here, the word ‘symbolic’ implies the explicit provision of high level symbols by some kind of meta-system. The level of description is also a key distinction — neuronal, behavioural, psychological, social etc.

This thesis is concerned with artificial systems, where learning occurs within the lifetime of the agent, and whose implementations tend towards the sub-symbolic end of the spectrum. The focus is on the 'neuronal' and behavioural levels of description, and on reinforcement learning techniques in particular. However, these distinctions are not intended to irrevocably carve up the domain space in such a way as to preclude overlap, and indeed many of these distinctions will be re-examined during the course of the thesis.

1.1 Supervised, unsupervised and reinforcement learning

Learning in artificial systems is often divided into *supervised* and *unsupervised* techniques. Supervised systems are trained using explicit input-output pairs where the task is to generalise over the training samples. In contrast, unsupervised learning algorithms are presented only with inputs, where the task is to model the underlying distribution of some data. A third class of techniques are described as *reinforcement learning*, where inputs are provided but no explicit target values are given. Instead, a *reward signal* provides feedback for explicit input-output pairs. The aim of reinforcement learning systems is the discovery, through trial and error, of a mapping from inputs to outputs that maximises this reward.

Reinforcement learning is sometimes subsumed under the supervised class because there is a teacher signal, albeit a weak one. Conversely, it is sometimes considered a special case of unsupervised learning since no target outputs are explicitly provided. However, reinforcement learning fully deserves its own category — partly because it does not fit comfortably into either of the other two, partly because of its separate bloodline, but mainly because it has its own clear and distinct theoretical foundations. The history and foundations of reinforcement learning are introduced in chapter 2.

Although this thesis is concerned with reinforcement learning problems, both supervised and unsupervised techniques will also provide a significant contribution.

1.2 Why use reinforcement learning

The key advantage of reinforcement learning is that the desired outputs need not be known in advance. This has obvious benefits when the details of a task or the agent's environment are unknown or poorly understood. Uncharted environments such as the surface of Mars might call for the ability of a robot to adapt itself rather than rely on pre-defined behaviour, or remote-control from earth. However, many environments closer to home may also be sufficiently poorly understood so as to benefit from autonomous adaptation. A celebrated example is found in (Tesauro, 1992, 1994) in which reinforcement learning is used to train a computer to play backgammon. Although the environmental dynamics are theoretically completely available, in practice the environment is rather less accessible because of computational constraints. Although a good strategy is easily recognised by the property that it wins games, actually finding such a strategy remains very challenging indeed. But this feature of being able to quantify the performance of a system without knowing beforehand how to achieve the best performance, is exactly what recommends a problem to a reinforcement learning solution. In the case of the Mars robot, the problem the designer is faced with is not knowing the details of the environment. In the case of learning to play backgammon, the environment is known but not understood, in the sense that with sufficient resources the task of winning games is trivial but, under practical constraints, appropriate generalisations are hard to find in advance. Another celebrated example is found in Crites and Barto (1996) in which reinforcement learning is used to improve a lift scheduling system involving a building with many floors and many lifts. Discovering an optimal schedule analytically is again infeasible, but such a policy can be approximated by a reinforcement learning system. As with the backgammon player, a suitable performance metric is readily available, this time in the form of the expected waiting times of lift users.

Adapting to unpredictable changes in an environment such as sensory-motor drift, or internal or bodily damage can also cause a designer problems, and this is another possible application of reinforcement learning. Providing performance can be measured inside the lifetime of the agent, reinforcement learning can be used to discover suitable behaviour. Here there is a clear analogy to the process of evolution, either natural or artificial, in which survival probability or a more general fitness function is used as a

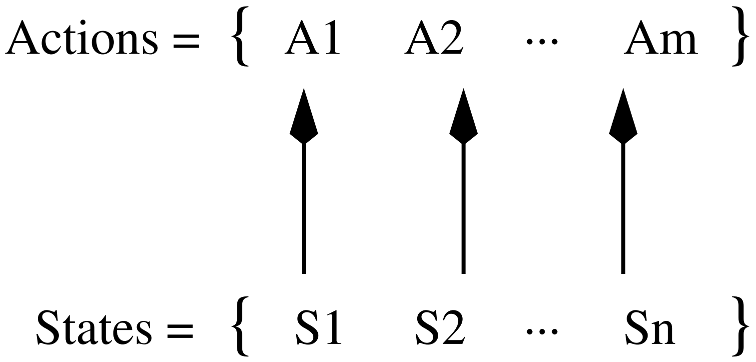


Figure 1.1: Summary of the basic reinforcement learning problem.

measure under which to optimise a system. The Backpropagation algorithm (Rumelhart et al., 1986), the Hopfield network (Hopfield, 1982), the EM algorithm (Dempster et al. (1977); see Bishop (1995) for a review) and many other neural and non-neural learning processes also utilise a scalar performance measure or *error* signal. Learning with respect to a scalar performance measure is common in a variety of in-lifetime, outside-lifetime, supervised, unsupervised, reinforcement learned, artificial and natural systems, and forms the basis of most learning techniques.

However, in the case of reinforcement learning, the price paid for removing the need for explicit training data is an increase in training time. Since the target outputs are not provided, a reinforcement learning algorithm must discover which actions are good and which are bad on the basis of trial and error.

1.3 Generalisation

The exact theoretical niche that has been carved out to form the reinforcement learning field can be summarised by figure 1.1. The basic problem consists of a discrete set of states, $\{S_1 \ldots S_n\}$, and a discrete set of actions, $\{A_1 \ldots A_m\}$. During training, and with the help of a scalar reward signal which provides external evaluation of any given state-action pair, the system must learn to produce the appropriate action for each state such that the overall reward received is maximised. The environment is assumed to be modelled by a Markov Decision Process (MDP). This is discussed shortly.

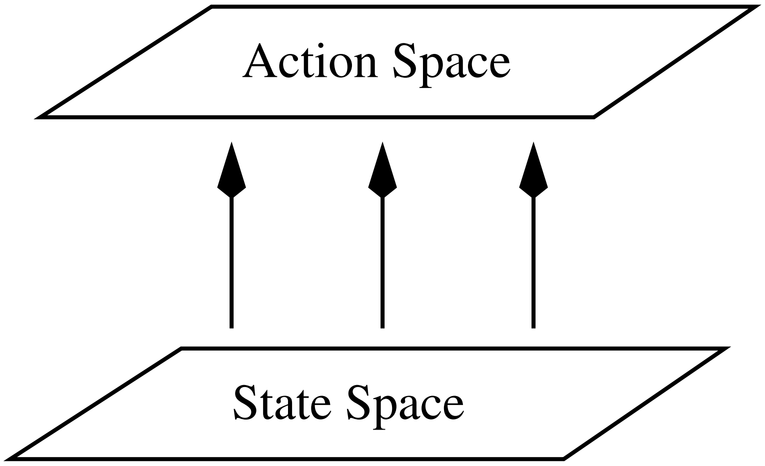


Figure 1.2: Summary of the more general reinforcement learning problem involving continuous state and action spaces.

Even in the case of a small number of states and actions this task may not be trivial since taking some actions in some states may yield little immediate reward, but take the system into a new state from which higher reward can be achieved at some future time. Since the aim is to achieve the highest reward over time, and not on any specific single occasion, this possibility must be considered. Also, the environment will generally be stochastic. This could mean that the reward yielded by a state-action pair will be represented by a distribution rather than a single value, but it could also mean that the state transitions occur with probabilities other than zero or one. This must also be modelled. The reinforcement learning theory, which is introduced shortly in chapter 2, succeeds under a number of assumptions in maximising the expected reward under all of these conditions.

However, in practice there are yet more difficulties since the number of states or actions could be very large or even infinite in the case of continuous spaces. A key consideration therefore in making the reinforcement learning theory usable is that of generalisation. The theory extends only as far as considering discrete states and actions, and so the system designer is still left with the often vital yet difficult task of selecting an appropriate generalisation technique. Hence the more general reinforcement learning problem can be stated in terms of figure 1.2.

Backgammon has about 10^{20} distinct board positions and so the backgammon player of Tesauro (1994) could not hope to represent each state explicitly. Similarly, the elevator problem of Crites and Barto (1996) was estimated to have 10^{22} distinct states so generalisation was a key issue here also. Any agent receiving real-valued input, or discrete input with added real-valued noise will, at least in theory, have an uncountably infinite number of states. In such cases it is often useful to consider how the problem can be transformed to adhere to the discretised template of figure 1.1. That the theory applies directly only to discrete spaces is unavoidable, since the reward function can only ever be sampled at a finite number of specific points and therefore its exact functional form will remain unknown.

1.3.1 Continuous action spaces

There have been many different approaches to state space generalisation including the use of neural networks, and some examples are considered in chapter 2. However, the usual approach to generalising over the *action space* is to hand-code a predefined (and usually small) set of discrete actions from which the learning system can choose. This simplification may be restrictive since the designer must suppose that he or she can guess a suitable set of actions beforehand. This is in conflict with the underlying principle of reinforcement learning which is to allow *autonomous adaptation* in order to maximise reward in uncertain or unknown environments. Some researchers, such as Gullapalli (1990), have addressed the issue of learning behaviour in continuous input *and* action spaces, and their work will be reviewed shortly.

In the meantime, the goal of this thesis can now be stated as: *The learning of optimal mappings between a continuous input space and a continuous action (or output) space, of arbitrary dimensions, in order to maximise a reward function that yields a scalar value for each sampled state-action pairing.* Although much effort has gone into researching the best methods for generalising over the input space, much less effort has been expended with respect to generalising over the output space. One key problem is that while the input distribution can be passively observed, the action space must be actively explored through a trial and error process. A reinforcement learning algorithm that deals with a continuous action space therefore essentially has two main problems — firstly that of collecting data from the action space, and secondly that of generalising over that data. In contrast, supervised learning only has the latter problem.

1.4 Motivation

The motivation for dynamic generalisation[1] of the input space is clear. The need for generalisation of the action space is less obvious however because it may appear that any continuous range of actions can be approximated to arbitrary accuracy by suitably interleaving a number of discrete actions. For example, consider a mobile robot attempting to learn to move to a stationary goal. At any point in time, the robot could need to turn in any real-valued direction in order to face the target, and hence the output space may be considered continuous. In practice however, the robot may get by with three discrete actions — a fixed angle left turn, a fixed angle right turn and the ability to move forwards. By sequentially combining these three behaviours appropriately, the goal could then be reached from any position. Furthermore, if actions can be switched quickly enough so that they can be made to interrupt and interfere with each other, then any continuous range of movement can be approximated. An example would be controlling the continuous temperature of a fridge with a thermostat that can only be on or off.

In general however, it may be very inefficient to approximate a continuous action with many discrete ones, particularly if there is latency in the system. If the robot can only change actions every few seconds for example, then a real-time controller might end up making a series of oscillating moves with only a small component in the desired direction. Consider a robot attempting to learn to push a box. If a new action can only be selected relatively infrequently, then the efficiency and even the eventual success of the process could be very sensitive to the action selected. Sharp turns could result in the box being lost before the next action has a chance to restore the balance, while shallow turns could result in inefficient control. Generating appropriately fine-tuned actions may be difficult with a set of pre-defined discrete actions.

[1] i.e. That which occurs during learning rather than being fixed beforehand as in Mahadevan and Connell (1991) for example. These two different approaches will be considered in more detail in section 2.10.

Consider a reinforcement system learning to sail a boat, say to a goal, by setting the relative angle of the boat and sail with respect to the wind and that goal. Here the range of possible actions in terms of these angles is continuous. Moreover, there may be a significant cost in correcting the boat, as well as a delay between taking an action and receiving reliable feedback. In this case we might prefer the system to be able to make accurate adjustments to the boat from a continuous range, rather than relying on frequent changes to the setup of the boat.

Some problems necessarily involve a one-off decision process that cannot be corrected later if it turns out to be wrong. An example that is used later involves learning to throw a ballistic projectile to a target, where the task is to select an appropriate throwing speed and angle. Once the projectile is launched, it may be that no corrections are possible. Learning to play any kind of sport involves a similar kind of ballistic continuous-action task, from learning to bowl a cricket ball to hitting a tennis stroke. Albus (1981)(pg. 170) talks briefly about the need for continuous actions in his *CMAC* learning model of the Cerebellar cortex. He concludes that the brain must be able to produce smoothly graded responses for behaviours such as jumping and applying appropriate forces to objects etc.

The latency issue is relevant for any application where speed is of the essence. A guided missile system cannot afford to make large changes during flight for similar reasons that a squash player cannot afford to approximate an optimal position on the court by a series of large oscillatory movements, or that the captain of a boat does not want to approximate a continuous course with excessively frequent discrete adjustments. We also might expect balancing under gravity to be more robust under smooth adjustments since discrete actions may lead to over-compensation and instability.

A final point is that even if a small set of discrete actions exist which can be used to approximate a continuous range of real-valued actions, this set may not be accessible to the designer a-priori. Increasing the set of discrete actions in the hope that at least some will turn out to be appropriate may lead to efficiency problems since the number of actions needing to be explored in order to discover the optimal ones will increase too.

The motivation for this thesis can now be stated as: *While abstract problems such as games can be played using an a-priori defined set of discrete actions, many real-world problems will require an adaptable set of real-valued actions drawn from a continuous range. Even where discrete approximations are possible, issues of speed, efficiency, latency, and reliability may recommend the use of adaptable real-valued actions.* It is this class of problems that this thesis is concerned with.

1.5 Delayed rewards

Apart from large or continuous state and action spaces, the other key difficulty facing practical reinforcement learning is that of *delayed rewards*. This refers to the difficulty of assigning credit to an action if the reward for taking that action occurs long after the event itself. As we will see, the theory itself has no trouble in dealing with delayed rewards, but rather it is in the efficient application of practical reinforcement learning algorithms that this issue becomes relevant. This thesis is not concerned with the problem of delayed rewards, although consideration will be given in appendix F to how efficient and appropriate state and action space representation may do something to ameliorate this particular problem.

1.6 Dynamic environments

Apart from the issues of stochastic environments, delayed rewards, and continuous state and action spaces, the issue of dynamic environments is also important. The theory deals with stationary environments, despite the reality that the task or the environment may change during learning. The impact of this possibility is considered at various points throughout this thesis, particularly in section 8.5.3.

1.7 Applications and emphasis

A number of specific learning problems are considered during this thesis, with each problem aimed at exploring a different aspect of a new model (introduced in chapter 5) with respect to the statement of intent on page 6. There are clearly a large number of dimensions open for exploring the reinforcement learning of real-valued mappings. For example, we could consider problems where the reward is immediate (as in Gullapalli (1990) for example), or where reward is significantly delayed (as in Tesauro (1994) for example). We could experiment with high dimensional state and action spaces, or concentrate on other scaling issues such as algorithmic efficiency, complexity, and compactness. Other aspects we might consider are non-stationary environments, and robustness to noise.

Many of the tasks will focus on the ability of the system to represent and generalise over a continuous action space, and for this reason will simplify many of the other problem dimensions mentioned above. In particular, we will often consider abstract regression style learning tasks in which the environment is stationary, the input and output spaces are of relatively low dimensionality, and the reward is immediate. This follows, for example, the approach of Gullapalli (1990). However, variety in the experiments performed for this thesis is intended to at least provide an indication of how the proposed model is likely to compare with existing approaches across a broad range of issues.

As an example, the question of noisy environments and partially delayed reward will be considered in chapter 5, in experiments performed in a robot simulator. In general, it is noted that robot learning provides a convenient framework for testing models of reinforcement learning because of the range of practical issues raised by problems from this domain. For example:

- **Speed and efficiency.** A robot must be able to acquire behaviours in a relatively small amount of time, particularly if there is a cost associated with learning.
- **Robustness to noise.** Unstructured environments generate noisy inputs to the learning system.

- **The need for generalisation.** Robot input and output spaces tend to have high dimensionality, and are often continuous, making generalisation an important consideration.

- **Non-stationary environments.** Real-world environments may change during learning, and a learning algorithm ought be able to adapt to this.

- **Non-equiprobable, interactively sampled input distributions.** Inputs to the system will often be generated through an agent's interaction with its environment, and therefore a suitable learning system must be able to cope with sequentially generated data that is drawn with bias from an implicit distribution.

- **Algorithmic compactness.** Huge computation and memory requirements may lead to violation of power supply constraints and are generally discouraged. Robots often benefit from being simple, small, and light.

- **Potential for parallelisation.** Roboticists are often particularly interested in reactivity, and long response or training times are usually to be avoided. Algorithms which lend themselves to parallelisation are expected to scale better in this respect.

The impression therefore is that a learning algorithm that works well in a robot learning domain will have implicitly addressed a number of key issues that ordinarily might be expected to obstruct practical implementation. Practicality is a key consideration throughout this thesis.

Other control problems are also considered throughout, usually to address a particular issue or subset of issues in detail. The robot learning problems of chapter 5 are chosen to address delayed rewards, and noisy, non-equiprobable environments, while more abstract regression style problems are used to discuss flexibility, efficiency, and representational issues. The problem of learning to throw a ballistic projectile (chapter 7) illuminates the discussion of learning multiple actions for the same state, and also provides an example of a task for which learning real-valued actions is actually necessary, rather than just desirable. This learning task is also used to uncover some key generalisation and scaling issues. The XOR problem is considered in chapter 8 for purposes of comparison with a key existing technique for learning and generalising over real-valued actions.

These applications are intended to provide a broad context of salient issues against which the performance of both existing and new techniques can be evaluated and compared. The emphasis is on practicality, applicability, and scalability, and these issues will constitute the focus of the thesis.

1.8 Thesis outline

A brief outline of the thesis is now given. Chapter 2 introduces the field of reinforcement learning, including a brief history of the subject and a review of the main theoretical components. Chapter 3 introduces some preliminary material on neural networks and robot learning, and chapter 4 concludes the introductory material with a review of existing techniques for the reinforcement learning of real-valued functions, with an emphasis on different approaches for representing the action space. The purpose of chapter 4 is also to discover a set of desirable properties of any reinforcement system addressing continuous state and action spaces. This set of properties will then be used to guide and evaluate the construction of a new model which is introduced in chapter 5.

The new architecture of chapter 5 is initially tested in a simulated robot domain, a simulated behaviour-based robot domain, and on various simple regression style problems. Chapter 6 then performs a preliminary analysis of the new model, and compares its performance in qualitative terms to the existing models discussed in chapter 4.

Chapter 7 introduces a new task which involves ballistic targeting. This tests the system under conditions where there may be infinitely many optimal actions for any situation, and considers how the system will represent such mappings.

Chapter 8 takes a prototypical continuous-space reinforcement learning model based on backpropagation and quantitatively compares it with the proposed model, which will be based on the self-organising map of Kohonen (1987). This chapter concentrates on the scaling potential of a distributed approach to representation compared with a local representation approach. We also consider the implications of non-stationary environments for both the distributed and local model classes. The discussion goes beyond the supervised/unsupervised distinction which is shown to be largely irrelevant for the purposes of generalisation in reinforcement learning. Here we are concerned with the most appropriate methods for generalising over continuous state and action spaces, and to this end a broad range of learning techniques may be adapted.

Chapter 9 begins with a discussion of some of the key issues pertaining to the applicability and scalability of the proposed model. The second half of this chapter discusses avenues for future work, including hybrid models that combine the self-organising map and the backpropagation algorithms, potentially bringing the best features of both model classes to bear on the problem. Some of these algorithms are built on the existing approaches to continuous-space reinforcement learning introduced in chapter 4. Chapter 10 concludes with a summary of the thesis.

Some additional work was also performed on the interactive abstraction of higher level categories. This work, which was inspired largely by Karmiloff-Smith's psychological account of child development (Karmiloff-Smith, 1995), is presented for completeness in appendix F. The work is presented as an addendum because although deemed interesting and potentially useful, it does not contribute directly to the thesis itself.

CHAPTER 2

Reinforcement Learning

In the following chapter, sections 2.1 through to 2.8 are essentially a review of the book: "Reinforcement Learning" by Sutton and Barto (1998). The book provides a lucid, comprehensive and consistent account of the theory and its history, and in the author's opinion represents the best introduction to the subject. This chapter also draws significantly on an authoritative reinforcement learning survey by Kaelbling et al. (1996).

The main topics of the thesis are now reviewed in this and the two subsequent chapters. This chapter is devoted to the history, theory and practical application of reinforcement learning, while the main purpose of chapter 3 is to introduce neural networks as an implementational paradigm. Both these sections provide the foundation for chapter 4 which concludes the introductory material by reviewing existing work on the reinforcement learning of real-valued functions, with an emphasis on different approaches to representing the action space.

2.1 History

Historically there are two main strands that contribute to the field of reinforcement learning: animal psychology and Dynamic Programming.

Animal learning is traced to Thorndike (1911), who suggested that an animal, given a choice of responses in a given situation, would when encountering that same situation again, be more likely to reproduce an action that resulted in satisfaction, and less likely to reproduce one that resulted in dissatisfaction. This intuitive idea was also developed by Pavlov (1927), and is commonplace in modern psychology.

In the late 1950s the phrase *optimal control* was used to describe a technique for minimising a measure of a dynamic system's performance. Bellman developed a functional equation — now called the *Bellman equation* — for calculating the value function of a dynamic system. The process of solving a set of these equations, either analytically or incrementally in order to first estimate the values of the various states of the system, and then derive a policy for maximising the expected return over the life of the system developed into the field of Dynamic Programming (Bellman, 1957), and today represents the theoretical grounding of all RL techniques.

It was not until the early 1980s that Barto, Sutton, Watkins and others began defining modern reinforcement learning, uniting the strands, clarifying the theory and, importantly, distinguishing the field from supervised learning, thereby giving RL its own identity and its own place in the machine learning literature.

2.2 Introduction

The intuition behind reinforcement learning (RL) is very simple — an agent learns for itself how to maximise a reinforcement signal from its environment by trial and error exploration of different actions in different situations. If the signal is designed to yield high reward at goal states, and low reward in situations that are to be avoided, then in learning to maximise that signal, the agent will hopefully also learn how to achieve its goals. Unlike supervised learning, where the desired output is presented along with the input, in RL only the *value* of an action is provided, and the agent itself is responsible for discovering and selecting appropriate actions based on the relative strengths of these values.

The standard example of a simple RL problem is the *n-armed bandit* where one of n levers must be pulled at each time-step, with each lever yielding a reward according to a fixed distribution. Imagine being in a situation with two levers and one hundred pulls to make. How would you maximise your reward? Clearly one strategy involves trial and error sampling until sufficient confidence is held in the belief that one arm yields a higher expected reward than the other, at which point only that arm should be pulled. Each arm could be tried just once, and then the one yielding greatest reward pulled thereafter, but this does not allow for an unlucky sample. If the reward distributions of the two arms are similar and you still have many goes left, it makes sense to take a larger number of samples (of both levers) to be sure you get the most out of the remainder of the game. Conversely, on the 100th go, the only sensible thing to do is pull the lever with the highest expected reward according to your experience so far.

Pulling the arm which is believed to yield the best result is known as *exploiting* the current knowledge. But in order to be confident about that knowledge, all options must first be *explored* (even if they initially appear likely to be worse) in case the new information uncovers greater reward in the long run. This is known as the *explore/exploit dilemma* since on the one hand exploration is necessary to uncover reliable information, but on the other hand exploitation is necessary to make the most of that information, and they cannot both be performed at the same time. It obviously makes sense to explore more at the beginning of a trial when the information gained will be of most use, and to exploit at the end when the cost of exploring will tend to outweigh the benefits of the new information gained. In most practical applications, optimal solutions to this dilemma are not known, but some commonly used strategies will be introduced shortly. For a discussion of bandit problems, see Narendra and Thathachar (1989).

The n-armed bandit problem is actually a special case of the more general RL problem, as there is only one state that the system can be in — namely that of being faced with pulling one of the arms. From this single state there are a number of actions with each action corresponding to pulling one of the arms. In the more general problem, the system can be in any one of a number of states with the optimal action depending on the state. An example is the game of noughts-and-crosses in which each board position can be thought of as a state of the problem, and in which there are nine actions — one for each square of the grid. Not all actions will be available in each state, and of course different actions will be preferred in different states. A system for selecting an action to take in each state is referred to as the *policy*. For example, one (unrecommended) policy for noughts-and-crosses would be to always take the top-left most square available.

2.3 Markov Decision Processes

It is convenient to consider the environment as a *Markov Decision Process* (MDP), an example of which is shown in figure 2.1. The system has four states (one of which happens to be a terminal state making it a *finite horizon MDP*), and two actions. The states are numbered 1 to 4, and the actions labelled a1 and a2. In each state, the two actions will take the system into a new state with a fixed probability which is indicated to the right of the colon for each transition. In the context of the RL problem, each *state transition* also yields a reward as a scalar value. It is easy to imagine the noughts-and-crosses example drawn and labelled like figure 2.1. In this case there would be $3^9 = 19683$ states[1], nine actions, $\frac{9!}{5!4!} \times 2 = 252$ terminal states, and the transition probability from each state under each action would be unity since the game is deterministic.

In the rest of this section, it is assumed that the environment is represented by an MDP. *This will imply that each state contains sufficient information so that the probability of moving to any next state, s', and receiving any reward, r, is the same given the current state and action information as if given the entire state-action-reward history of the environment*. This can be expressed by the following equality, which defines the *Markov property*:

[1]Not all of these would be valid board positions.

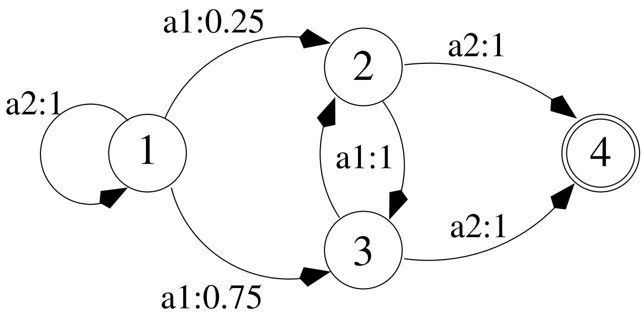


Figure 2.1: A simple Markov Decision Process consisting of four states and two actions.

$$P(s_{t+1}=s', r_{t+1}=r|s_t,a_t) = P(s_{t+1}=s', r_{t+1}=r|s_t,a_t,r_t,s_{t-1},a_{t-1},r_{t-1},\ldots,s0,a0) \tag{2.1}$$

for all s_{t+1}, r_{t+1}, and state-action-reward histories, where s_t is the state at time t, a_t is the action taken at time t, and r_t is the reward received in moving to state s_t.

This assumption ensures that at each state, the agent has sufficient information to make a perfectly informed decision given the boundaries of the particular problem. In the noughts-and-crosses example, coding the board positions as states results in a game with the Markov property because a complete board position contains all the salient information for winning a game. But coding a maze with states corresponding to "left corner", "right corner", "corridor" does not yield an MDP because escaping a maze requires at least an implicit knowledge of location which cannot usually be inferred from the immediate surroundings. However, if a state history is maintained so that the escapee can remember the types of previous junctions, then it may be possible to localise, but this then corresponds to a different MDP in which the states are n-tuples of the old "left/right/corridor" states. Of course if the states are coded as *particular* corners and corridors, then it *is* an MDP because there is as much information in a single state as in an entire state history (with respect to escaping the maze). The maze is an example of a *deterministic* process since one assumes transitions between states will occur with probabilities zero or one (although of course one could easily contrive an example where they do not). An example of a more general, non-deterministic MDP is encountered shortly in figure 2.2.

Note that even though an agent may have access to sufficient information so that the Markov property *is* satisfied, an inability to perceive that information (perhaps through impoverished sensing apparatus) may lead to the agent effectively facing a non-Markovian decision process. This is referred to as *perceptual aliasing*.

2.4 The basics

The standard reinforcement problem is defined using the following elements:

- **Set of states**
  A set of discrete and distinct states, $\mathbf{S}$, corresponding to the learning agent's perception of the states of its environment. A state could be a board game position, a vector of robot sensor readings, a position within a maze etc.

- **Set of actions**
  A set of discrete actions, $\mathbf{A}$, available to the agent. Not every action need be available in every state.

- **Policy**
  The policy, π, dictates which actions are to be taken in each state. Policies may be stochastic.

- **Reward function**
  The real-valued reward function, $\mathbf{R}$, maps states, state-action pairs or state-action-state tuples to reward values. Reward values may be positive, negative, or zero indicating no reward. The reward function is usually unknown to the agent, and must first be explored and then exploited.

- **Value function**
  The value function, $\mathbf{V}$, is a central idea to RL techniques and maps each state to a measure of the value of that state. The value of a state is taken to reflect the expected accumulated reward from that state on. V is usually taken to refer to the *actual* value function, while $\hat{V}$ refers to the *estimated* value function. V^{π}

refers to the value function under some policy, π, while V^* refers to the optimal value function — i.e. the value function under the optimal policy.

- **Model of environment**
  The environment model, which may or may not be known to the agent, predicts the behaviour of the environment by mapping state-action-nextstate tuples to probabilities. The environment model is provided by $T(s,a,s')$ which returns the probability of moving to state s' after taking action a in state s, for all s,a,s'.

  The environment model is provided in the form of a transition function, T, from state-action-nextstate tuples to probabilities.

This outlines the basic RL context that was introduced in figure1.1. The additional constraint has been added that the environment model behaves as an MDP.

At each time-step an agent moves from one state to another by taking one of its available actions, and in so doing receives a scalar reward. The question is, against what measure should the agent's behaviour be optimised? One answer is to attempt to maximise the sum of all expected future reward, up to a *receding finite horizon*:

$$E\left(\sum_{t=0}^{h} r_t\right) \tag{2.2}$$

where h is the horizon and r_t is the reward received from the environment at time t after an action is taken.[2] [3] This *return* has to be 'expected' because of the stochastic nature of the environment. It is not assumed for example that an action guarantees a particular state transition, only a probability of that transition. If a task is of finite length, as with the two-armed bandit example earlier, then this approach may be adequate, but since

[2] An alternative is the *fixed* finite horizon in which the reward is summed all the way up to the fixed end of the trial.

[3] Strict statistical notational convention dictates that uppercase R is used to denote the random variable representing reward inside an expectation. However, for consistency, the notation adopted here and throughout is that of Kaelbling et al. (1996) and Sutton and Barto (1998).

we may not wish to make such an assumption, a more common value to attempt to maximise is the *discounted return*:

$$E\left(\sum_{t=0}^{\infty}\gamma^t r_t\right) \tag{2.3}$$

where $0 \leq \gamma < 1$ is called the *discount factor*. The idea behind (2.3) is that rewards are exponentially decayed as they become more and more distant and this ensures a finite sum, even on an indefinitely long training episode. There is also an intuitive appeal in trying to maximise immediate reward more than distant reward. In this way, γ effectively sets the horizon.

The value function, V^π, is defined as (2.2) or (2.3) for each state, based on the information provided by the reward function following that state given a policy, π. The aim of reinforcement learning is to discover an optimal policy, π^*, which maximises (2.2) or (2.3). If the environment model is known explicitly in terms of the transition probabilities and the reward function, then it may be feasible to analytically solve for the value function under the optimal policy, to give first V^* and then π^*. However, in many cases the environment model is not known, and a solution must be approximated by an iterative sampling method. This thesis is concerned exclusively with problems where the environment is not known.

2.5 Dynamic Programming

Assuming that (2.3) is the value we wish to maximise, and therefore first estimate, the value function, V, can be expressed by the Bellman equation (Bellman, 1957):

$$V^\pi(s) = \sum_{a\in A}\pi(s,a)\sum_{s'\in S}T(s,a,s')\left[R(s,a,s')+\gamma V^\pi(s')\right] \tag{2.4}$$

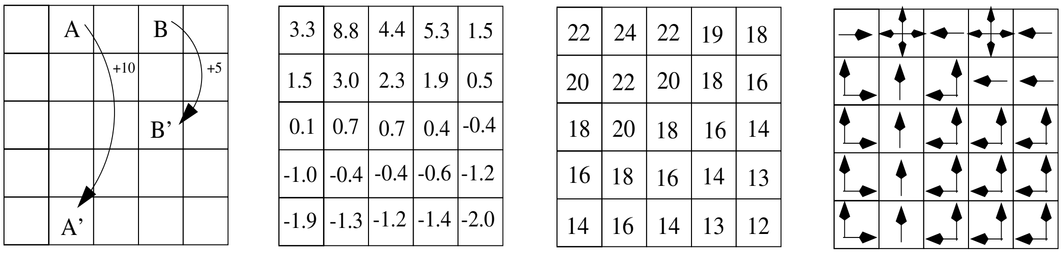
where $\pi(s,a)$ is the probability of taking action a in state s under policy π, $T(s,a,s')$ is the probability of s' being the successor state of s following action a (i.e. the state-transition function), and $R(s,a,s')$ is the reward elicited from the environment by taking action a in state s and ending up in state s'.[4]

The Bellman equation, which can be viewed as a recursive definition of equation 2.3, asserts that the value of state s under policy π is the result of summing, for each action and each possible successor state, the expected reward of that transition plus the discounted value of that successor state.

For a suitable RL problem, this yields a set of simultaneous equations, one for each state, which can be solved to yield the value function V^{π}. The following example is taken straight from Sutton and Barto (1998) (pg 71): Consider the grid world of figure 2.2a in which each square is a state from which the agent may choose one of the following actions: “up”, “down”, “left” or “right”. Each of these actions takes the agent to the appropriate neighbouring state and yields no reward except that attempting to move off the grid results in no movement and a reward of -1, and any action taken in states A or B results in a move to A' or B' with rewards of +10 and +5 respectively. Figure 2.2b shows the value of each state, as calculated by equation 2.4, for the policy in which each action is equally likely in each state, and with the discount factor, $\gamma = 0.9$. The negative values of edge squares in the lower half of the grid reflect the probability of the agent stumbling off the grid at these points. States A and B have high values, as do their neighbours, because of the potential for achieving the +5 or +10 rewards. However, the value of state A is slightly diminished by both its proximity to the lower edge of the grid via the special state transition $A \rightarrow A'$, and also the distance of the inevitable successor state, A', from the rewarded states A and B.

Each value is the expected discounted reward from that state onwards for the equiprobable policy. However, what we are more interested in is the *optimal policy*, π^*, which guarantees the greatest possible future reward. Equation 2.5 shows the *Bellman optimality equation* for V^*, which yields the expected return of each state if the best possible action is always taken. In the same way as before, a set of simultaneous equations

[4]We use R for the reward function (from state-action tuples), and r_t to denote the reward at a particular time, t.



a) Grid World b) $V^{\pi_{\text{Equal}}}$ c) V^* d) π^*

Figure 2.2: Application of the Bellman equations. Reproduced with permission from (Sutton and Barto, 1998)(pg. 170).

— one for each state — can be solved to yield V^*.

$$V^*(s) = \max_a \sum_{s' \in S} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right] \tag{2.5}$$

This equation is very similar to the previous Bellman equation except that instead of considering all possible actions from state s, only the action that maximises the future return is used. Figure 2.2c shows the V^* values for each state calculated using (2.5), and figure 2.2d shows the optimal policy, π^*, which can be generated by always selecting an action that maximises the right hand side of (2.5) for the current state. The optimal policy happens to prescribe moves that takes the system into state A as quickly as possible (unless avoiding state B in the process would require a detour).

However, solving n simultaneous equations in n unknowns where n is the number of states scales with $O(n^3)$ and soon becomes too expensive. *Dynamic Programming* alleviates this problem by changing the Bellman equation into an update rule that can be applied iteratively, one state at a time:

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right] \tag{2.6}$$

where V_k is the value function at the k^{th} iteration. Note that V_0 should be initialised to finite values.

For some policy, π, V_k is updated at every state using the previous values of V_{k-1}. The reason successive approximations improve the accuracy is that fresh information is being injected by the term $R(s,a,s')$ during each iteration. It can be shown that V_k converges to V^π as $k \to \infty$ (Bellman, 1957), and calculating V^π by iteratively updating the value function in this way is called *iterative policy evaluation*.

Now, based on V^π, it is possible to improve the policy to take advantage of the approximated value function. For example, in figure 2.2(b), knowing $V^{\pi_{Equal}}$ suggests a number of improvements to π in which the higher value states are preferentially sought. In practice, this can be achieved by setting $\pi(s,a) = 1$ for the $a \in A$ that maximises $\sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^\pi(s')]$, and setting $\pi(s,a) = 0$ elsewhere. This is called *policy improvement* and is guaranteed to yield a better policy, π', if one exists.

But this now means that the value function is based on an out of date policy and needs to be recomputed to reflect the new improved policy, π'. In this way, by repeatedly performing iterative policy evaluation followed by policy improvement, better and better policies are found converging on the optimum policy, π^*, and a corresponding optimum value function V^*. This is called *policy iteration*, and forms the theoretical basis of all practical RL techniques.

Equation (2.6) was an iterative version of the Bellman equation of (2.4). Similarly, the Bellman *optimality equation* of (2.5) can also be expressed as an iterative update rule:

$$V_{k+1}(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right] \tag{2.7}$$

which also directly converges to V^* without the need to maintain an explicit policy. This corresponds to the previous update rule in which the policy is effectively updated immediately rather than waiting for policy evaluation to converge. It is also possible to update $V_{k+1}(s)$ on states in any order, and providing each state is continually visited and never left unchanged indefinitely, convergence is still guaranteed as $k \to \infty$.

The principle behind policy iteration is that information about all rewards is passed around the system so that the value of each state eventually accurately reflects its intrinsic worth to the agent with respect to the expected return.

2.6 Monte Carlo techniques

Dynamic Programming requires that a complete model of the environment be known in terms of the state transition probabilities, $T(s,a,s')$, and the reward function, $R(s,a,s')$. In practice this information is unlikely to be available, and so the *Monte Carlo* method is introduced.

Trials are now required to be finite, so a guarantee is required that a terminal state of the MDP will be reached sooner or later. As with Dynamic Programming, the Monte Carlo approach aims to generate increasingly accurate estimates of the value function. Unlike Dynamic Programming however, where $V(s)$ is recursively updated using the value function at other states, Monte Carlo techniques update $V(s)$ towards the actual reward received from state s until the end of the trial. So the value function for a given policy, π, can be written as:

$$V^{\pi}(s_T) = E\left\{ \sum_{t=T}^{\text{trial end}} r_t | \pi, s_T \right\} \tag{2.8}$$

where T is the time at which state s is first encountered. Note that because the trial is restricted to being finite, (2.2) is now being used instead of (2.3) as the quantity to maximise. However, it is still common to discount:

$$V^{\pi}(s_T) = E\left\{ \sum_{t=T}^{\text{trial end}} \lambda^{t-T} r_t | \pi, s_T \right\} \tag{2.9}$$

$V^{\pi}(s)$ can be calculated by running N trials, and for the first visit to state s (at time T) during trial k, calculating the value:

$$\mathcal{R}_k = \sum_{t=T}^{\text{trial end}} r_t \tag{2.10}$$

Then (2.8) is approximated by the *Monte-Carlo first visit estimate*:

$$V^{\pi}(s) \approx \frac{\sum_{n=1}^{N} \mathcal{R}_n}{N} \tag{2.11}$$

with convergence as $N \longrightarrow \infty$.

But there are two important implications of not having an environment model. The first is that the value function is no longer sufficient for finding an optimal policy because even if the agent knows which the best states are, it does not know how to get there from the current state without the state transition function, $T(s,a,s')$. For this reason, instead of estimating the value of states using $V(s)$, estimates are made of the values of *state-action pairs* using the *action value function* (also *action* function), $Q(s,a)$.[5] The theory is the same as before with $\lim_{trials \to \infty} Q^{\pi}(s,a)$ =E(Total reward from taking action a in state s to the end of that trial, under π) except that now a state transition function is implicitly built into the action value function. Making a policy optimal with respect to Q^{π} is now simply a matter of always choosing the action that maximises $Q^{\pi}(s,a)$ at each state s. This is known as a *greedy policy*. This then becomes π', which in turn is evaluated by $Q^{\pi'}$ and so policy iteration continues in the usual way.

However, the second important implication of not having an environment model is that the issue of exploration must now be addressed. The agent is now responsible for sampling its own environment, whereas before the environment details were provided explicitly. So now, rather than updating the policy so that the action that maximises

[5]This formulation pre-empts Q-learning, which is introduced shortly.

$Q^{\pi}(s,a)$ is *always* chosen, π' is instead formed by *usually* choosing the action that maximises $Q^{\pi}(s,a)$ while occasionally selecting one of the other actions. This balances exploration and exploitation. These are known as ε-*soft* policies because each available action has a non-zero probability of being taken. One common ε-soft policy is to select the currently preferred action with probability $1-\varepsilon+\frac{\varepsilon}{|A|}$ and all other actions with probability $\frac{\varepsilon}{|A|}$, for some small value of ε. This is known as an ε-*greedy* policy. A similar but smoother approach is to select actions according to a Boltzmann distribution of their corresponding action values:

$$P(a) = \frac{e^{Q(s,a)/T}}{\sum_{b \in A} e^{Q(s,b)/T}} \tag{2.12}$$

With ε-soft exploration, convergence to the optimal policy is once again assured providing the policy converges to pure greedy. This is easily achieved by reducing ε or the temperature parameter, T, to zero. This ensures a shift from exploration to exploitation.

The advantage of Monte Carlo techniques of not requiring an environment model will turn out to be decisive not only when the environment is unknown, but also when $T(s,a,s')$ or $R(s,a,s')$ are known implicitly but difficult to calculate explicitly. See Sutton and Barto (1998)(pg 113) for an example. A disadvantage of having to implicitly model the environment by making the domain of the action value function state-action pairs rather than just states, is that the action function must now be stored and updated at many more indices (by a factor of $|A|$).

2.7 Temporal Difference learning

Like Dynamic Programming, *Temporal Difference* methods (Sutton, 1988) update the value function based recursively on other estimates, making the approach suitable for infinite horizon tasks and on-line, interactive learning. But like Monte Carlo methods, no model of the environment is necessary. Temporal Difference learning thus captures the best of both worlds, and for this reason dominates the standard account. As usual, the aim is to estimate the value of a state in terms of (2.3).[6] Temporal Difference learning is so called because $\hat{V}_t(s)$ (note that we now use $\hat{V}$ because we are now dealing with an estimate of the value function.) is updated based on the difference between $\hat{V}_t(s)$ and $\hat{V}_t(s')$, where s' is the state encountered immediately after s. The basic Temporal Difference update rule is:

$$\hat{V}_{t+1}(s) = \hat{V}_t(s) + \alpha\Big[\{r + \gamma\hat{V}_t(s')\} - \hat{V}_t(s)\Big] \tag{2.13}$$

which is applied immediately after receiving reward r for moving from state s to s'. The expression in the curly brackets corresponds to the contents of the square brackets in (2.7), and represents the target of the update. This is just a recursive formulation of (2.3). The rest of (2.13) moves the current estimate $\hat{V}_t(s)$ towards this target by an amount proportional to the *learning rate*, $0 < \alpha < 1$.

Providing each state is continually visited under some policy, π, and the learning and exploration rates are annealed to zero according to the constraints of (2.14), then $\hat{V}(s)$ will converge on the familiar return of (2.3) for that state, and therefore $\hat{V}$ will converge to V^π. In essence, this is the approach used in the backgammon player of Tesauro (1994).

$$\int_{t=0}^{\infty} \alpha(t) = \infty \text{ and } \int_{t=0}^{\infty} \alpha(t)^2 \leq \infty \tag{2.14}$$

2.7.1 Sarsa

As has already been seen with the Monte Carlo method, if an environment model is not available, the value function, V, is insufficient for improving the policy. Therefore,

[6] Equation (2.3) is being used again because non-finite MDPs are now being considered.

in the *Sarsa* algorithm of Rummery and Niranjan (1994) the action value function, Q, is again requisitioned to give the Temporal Difference update rule:

$$\hat{Q}^{\pi}_{t+1}(s,a) = \hat{Q}^{\pi}_{t}(s,a) + \alpha\Big[\{r + \gamma\hat{Q}^{\pi}_{t}(s',a')\} - \hat{Q}^{\pi}_{t}(s,a)\Big] \tag{2.15}$$

where a' is the next action to be performed from state s' according to the current policy, π.

The theory is a simple extension of Dynamic Programming, based on Bellman equations for Q^{π} and Q^{*}.[7] Following the discussion so far, we can see that the repeated update of $\hat{Q}^{\pi}(s,a)$ towards $(r + \gamma\hat{Q}^{\pi}(s',a'))$ based on sample experience will yield (2.3), for the current policy.

Given Q^{π}, the policy can then be improved to exploit this information in exactly the same way as the Monte Carlo method — by choosing the action a in state s that maximises $Q^{\pi}(s,a)$. Through policy iteration, Q^{π} converges to Q^{*} and π to π^{*}, providing as usual that the environment is modelled as an MDP, the learning rate satisfies (2.14), all states are visited infinitely often in the infinite limit (using an ε-soft policy for example), but that exploration is eventually reduced to zero (see Singh et al. (2000) for convergence proof). Note that following the discussion of policy iteration, there is no need to wait for $\hat{Q}^{\pi}$ to converge on Q^{π} before updating π. In fact, here π is effectively updated after every single update to $\hat{Q}$ simply because π is based on the current $Q - values$. This particular Temporal Difference method is called Sarsa because the update rule uses s,a,r,s' and a'(Sutton, 1996). This and the following technique are referred to as *bootstrapping* because, unlike Monte Carlo, estimates of expected return are updated largely towards other estimates which themselves are based on further estimates etc.

[7] See Sutton and Barto (1998) for these equations. They are similar to the Bellman equations already encountered, and do not add anything to this particular discussion.

2.7.2 Q-Learning

Sarsa is actually a minor and recent adaption to one of the most theoretically important, and most popular RL methods known as *Q-learning* (Watkins, 1989), which uses the update rule:

$$\hat{Q}^{\pi}_{t+1}(s,a) = \hat{Q}^{\pi}_{t}(s,a) + \alpha\Big[\{r + \gamma \underset{a'}{max} \hat{Q}^{\pi}_{t}(s',a')\} - \hat{Q}^{\pi}_{t}(s,a)\Big] \tag{2.16}$$

This is identical to Sarsa except that when considering the next state-action transition, the action a' is chosen that will *maximise* the next Q-value as opposed to choosing a' according to the current policy. This means that the policy being *evaluated* is closer to the optimal policy for the current Q-function (i.e. with no exploration) even though the policy being used for *control* may still be involved in exploration. Sarsa effectively models its own exploration as part of the dynamics of the environment, while Q-learning does not. Modelling the exploration may be useful if such exploration can profoundly affect the reward (see Sutton and Barto (1998), page 150 for an example).

Q-learning is shown to converge to an optimal policy under the usual assumptions (Watkins and Dayan, 1992), and it remains the most popular reinforcement learning algorithm because no model of the environment is required, it is intuitive, easy to implement, and can be run interactively with updates made immediately, as and when states are visited. These features make the algorithm suited to a wide variety of learning tasks. For example, Araujo and Grupen (1996) use Q-learning in a foraging task to map states to high level behaviours which are generated beforehand. Digney (1996) uses *nested Q-learning* to build hierarchical control structures for use in a grid-world environment. A particularly celebrated example of this Temporal Difference method is found in Mahadevan and Connell (1991), where a robot learns to find and push boxes within a behaviour based framework. Q-learning is also employed in Crites and Barto (1996), where an extension of the algorithm is used to discover a policy for efficiently dispatching lifts to minimise waiting times.

2.8 TD(λ)

To round the theory off neatly, the Monte Carlo and Temporal Difference methods can be shown to be special cases of a more general formalism, $TD(\lambda)$ (Watkins, 1989).

In Monte Carlo methods, the value of each state is updated towards the actual reward received from the first visit to that state to the end of the episode. In the Temporal Difference algorithm, the value function is estimated recursively in the sense that it is updated towards the immediate actual reinforcement plus the discounted *estimated* value of the next state (or state-action pair). TD(λ) is a more general algorithm which provides smooth control over the degree to which actual returns and estimated returns are blended to produce the target towards which the value function is updated.

Recall that in (2.13), $\hat{V}(s)$ was updated towards the *1-step corrected return*, but just as plausible are the 2-step, 3-step or n-step returns:

$$\begin{aligned}
&\text{1-step return} = r + \gamma\hat{V}(s') \\
&\text{2-step return} = r + \gamma r' + \gamma^2\hat{V}(s'') \\
&\text{3-step return} = r + \gamma r' + \gamma^2 r'' + \gamma^3\hat{V}(s''') \\
&\vdots
\end{aligned}$$

where $s, s', s'' \ldots$ is the sequence of states as they are visited, and $r, r', r'' \ldots$ is the sequence of rewards received on entering these states. If the trial length is finite, and n large enough to reach the end of each trial, then the n-step return is just a non-bootstrapping target as used in the Monte Carlo algorithm. Hence there exist a range of methods with Monte Carlo at one extreme and basic Temporal Difference at the other. It is a simple matter to combine these two extremes in a continuous manner by updating $\hat{V}(s)$ towards a weighted sum of n-step returns:

$$R_1 + \lambda R_2 + \lambda^2 R_3 + \ldots + \lambda^{n-1} R_n \tag{2.17}$$

for $0 \leq \lambda \leq 1$ (note the distinction between λ and γ!), where R_m is the m^{th}-step return from state s. Since the weights of the n-step returns should sum to unity (for $n = \infty$) in order to respect the estimate of (2.3), an appropriate normalisation factor is introduced so that (2.17) becomes:

$$(1-\lambda)R_1 + (1-\lambda)\lambda R_2 + ... + (1-\lambda)\lambda^{n-1}R^n \tag{2.18}$$

The term, λ, is the continuous parameter referred to in $TD(\lambda)$, which in its limits of zero and one represents 1-step Temporal difference and Monte Carlo methods respectively. Although this may seem like a rather contrived way of combining actual and estimated returns, it actually represents the theory underpinning an intuitively appealing and popular set of algorithms defined by the use of *eligibility traces* (Watkins, 1989). Such algorithms maintain a record of recently visited states and use this history to accelerate the passing of reward information across the value or action function. Versions of this algorithm also exist for Q-learning and Sarsa in the form of $Q(\lambda)$ (Watkins, 1989; Peng, 1993; Peng and Williams, 1996) and $Sarsa(\lambda)$ (Rummery, 1995) respectively. See Tesauro's backgammon player (Tesauro, 1992, 1994) for an application of TD(λ). See Sutton (1996) for an application of Sarsa(λ) and Araujo and Grupen (1996) for an application of Q(λ) to simulated robot control.

Although there is no principled analysis available, Sutton (1996) concludes that $0 < \lambda < 1$ is likely to be optimal with $\lambda = 0$ and $\lambda = 1$ empirically performing relatively poorly. In his backgammon application, Tesauro reports that: "...λ appeared to have almost no effect on the maximum obtainable performance, although there was a speed advantage to using large values of λ [corresponding to Monte Carlo]". Jaakkola et al. (1994), amongst others, have provided a convergence proof for $TD(\lambda)$.

Although a number of variants and extensions to the above algorithms have been proposed, the previous section provides as much history and theoretical background as is interesting and relevant to this thesis. The reader is referred to Sutton and Barto (1998) and Kaelbling et al. (1996) for a more thorough treatment. The focus now moves from the theory to the practice.

A brief notational comment is required at this stage. In the remainder of this thesis when we talk about 'Q-values' we will be referring to the *estimated* Q-values — i.e. the function, $\hat{Q}$. However, to simplify notation, we will omit the superscript and refer to estimates simply by using the function, Q. We will also adopt the simplifying notational convention of omitting the policy superscript, since there will always be an implicit assumption that we are estimating expected return for the current policy, and not the optimal policy. Furthermore, the term 'Q-value' will be used to refer to any estimate of expected return for state-action pairs.

2.9 Practical reinforcement learning

The theory provides the following: An iterative, incremental and interactive method that guarantees convergence of the value function, V, to either (2.2) or (2.3), under the assumptions that the environment is modelled as an MDP, every state is continually visited, and the learning and exploration rates are annealed appropriately. The value function, V, estimates the values of *states* of the MDP, which requires that the basic value function update rule (2.13) makes use of an explicit environment model. If the environment model is unknown, then the value function, V, is replaced by the action function, Q, which estimates the value of each state-action pair. Now the environment model is implicitly learned as part of the action function. By interleaving *policy evaluation* and *policy improvement*, an optimal policy is guaranteed to be found.

2.9.1 The assumptions

In practice the MDP assumption can rarely be met, since in many applications the sensory information fails to uniquely identify the state of the environment. This problem of *perceptual aliasing*, in which states are confused with each other, is exactly why escaping a maze is difficult, even for us. In general, the complexity and uncertainty of the real-world will make it impossible to satisfy the MDP assumption. Also note that this assumption is left unsatisfied when the environment is modelled as an MDP, but when this model is dynamic. If the state transition probabilities and reward function change over time, as may well be the case in a real-world problem, then convergence to such a moving target cannot be guaranteed.

The assumption that each state (or state-action pair) is continually visited can be satisfied by always maintaining an appropriate amount of exploration (as discussed in section 2.6). The convergence proof effectively requires that each state is visited an infinite number of times. In practice, trials must be of finite length, and some states may suffer particularly from under exposure, resulting in inaccurate value estimates. However, since the algorithms outlined above are interactive or *on-line*, the most frequently visited states will conveniently tend to have the most accurate value estimates.

The assumption of appropriate learning and exploration rates is easily satisfied in the limit of infinite trial length by the conditions of (2.14). However, in the finite case, finding a suitable set of learning rates that maximise performance is an empirical challenge.

Having established that, in practice at least, the criteria for convergence cannot be satisfied, the question now arises as to how well these algorithms perform when the assumptions are not met. Happily, the answer appears to be quite well. By choosing a suitable state representation the task can be made as close to an MDP as possible. Incorporating previous sensory data can also help to reduce the problem of perceptual aliasing. Judicious selection of the exploration rate can allow the assumption of continuously visited states to be at least partially satisfied, and in any case, the interactive nature of the algorithm suggests that accuracy will tend to reflect the exposure and therefore the relevance of different parts of the environment. The empirical selection of a suitable set of learning parameters also seems to be reasonably straightforward in the majority of cases. In addition, some evidence has been presented that other parameters, such as λ, may not have a huge impact on performance, and that satisfactory if not optimal values will be easy to find.

2.9.2 Delayed rewards

Mataric (1997) identifies two main problems that need to be addressed in reinforcement learning. The first is that of delayed rewards, or more generally, credit assignment. Although the theory provides a guarantee of optimality for infinite length trials (or an infinite number of finite trials), many practical applications, such as those involving physical robots for example, may be very restricted with respect to the number of environment samples that can be made. For this reason it is important that reward information propagates around the value function as quickly as possible. As an illustration of the problem, in Tesauro's backgammon application, TD-Gammon, the reinforcement of all states was zero except for the final state of a won game at which point the reward was one. This is beautifully simple, and requires a minimum amount of prior game knowledge, but hundreds of thousands or millions of complete games were needed to allow the reward of won games to propagate back to the early game states.

Mataric (1994, 1997) addresses the issue of delayed rewards by introducing *progress estimators* which provide a handcrafted continuous reward function which augments the reward information that is received at goal states. For example, a progress estimator might provide an estimate of the distance of an agent from a goal in a robot navigation problem. Progress estimators address the more general RL aim, identified by Kaelbling et al. (1996), of making the reward signal as local as possible. Breaking a task up into subtasks or a control system into behaviours, with each task or behaviour having its own reward function, is another approach to reducing the time between an action being taken and reward for that action being received (see Mahadevan and Connell (1991); Mataric (1994, 1997) for some examples). Tesauro's backgammon player exemplifies the problem of delayed rewards since the only information from the environment always comes on transition to a terminal state of the MDP. Attempting to provide as rich a reward signal as possible is an important part of encoding a task for an RL solution. Caution is advised though. Supplying handcoded intermediary signals to *shape* the learning process may result in the wrong behaviour being accidentally reinforced. Note that in the backgammon example, maximising the (rather weak) reward signal was *guaranteed* to maximise playing ability given enough training time. But consider what might have happened by 'enriching' the reward signal by a progress estimator designed to reward intermediate game positions that were mistakenly judged by an expert to be strategically advantageous.

Although this thesis is not directly concerned with delayed rewards, the issue is encountered at a number of points throughout the thesis. It is introduced here largely for completeness.

2.9.3 Large state spaces, and generalisation

The second of the two major problems facing RL application identified by Mataric (1997) is the possibility of large or continuous state or action spaces. Consider again the backgammon example, in which there are about 10^{20} distinct board positions, and therefore the same number of states. Representing the value of each board position explicitly is clearly impossible, so Tesauro used backpropagation to train a neural network to approximate the function, V. Note that because the environment model is known (in terms of the available successors to the current state), the value function V is sufficient for learning an optimal policy. The algorithm used is actually $TD(\lambda)$, but (2.13) also characterises the approach. However, at each state, instead of a table entry for $V(s)$ being updated, the network is trained towards the pair (I,O), where I is the input vector corresponding to the current board position, s, and O is the target inside the curly brackets (of 2.13). Because of the way the experiment was set up[8], the value function approximated by the network effectively mapped each board position to the probability of winning from that board position. Optimising the policy was then achieved by simply selecting the move from a list of legal possibilities that lead to the state with highest $V(s)$ according to the network. No exploration was built into the move selection because the dice throws themselves were considered to generate enough noise[9]. The system was trained against itself, so a large number of games could be played.

Generalising over the state space using a non-linear function approximator yielded excellent results for TD-Gammon, with the system learning to play backgammon to club standard. By hand-coding salient board features into the representation at each state, performance was improved to rival the world's best players. A pleasing addendum is that some of the opening plays learned by TD-Gammon went on to change the opening theory of the game.

[8]No discounting took place and recall that the reward was zero in all states except won positions, where it was one.

[9]The reader could refer to http:/www.funcom.com/lang_en/lang_en/games/backgammon/rules.html for an explanation of the game.

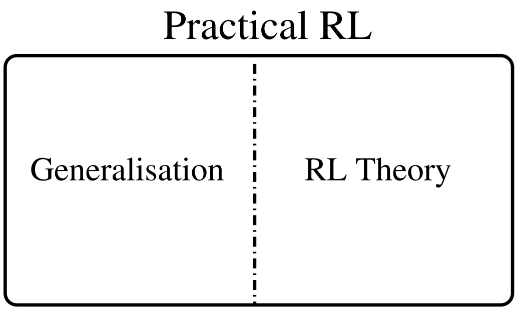


Figure 2.3: The independent relationship between RL and Generalisation theory. The diagram emphasises the point at which RL finishes, and where the responsibility for scaling RL primarily resides. In particular, a lack of ability in dealing with large or continuous state spaces is not an inherent problem of the RL theory itself. The latter provides a clear set of results within a clearly defined set of boundaries. The sophistication and potential applicability of RL is not limited by the theory, but by the techniques with which the theory is combined.

However, a fourth criterion for guaranteed convergence that has only been implicit so far in this chapter, is that the value or action function be represented explicitly as a lookup table. This drives another wedge between RL theory and its practical application since large or continuous state spaces will in general preclude this condition being met. Effective generalisation over large or continuous state and action spaces is the key issue facing real-world applications of RL, and this is the main focus of this thesis, particularly with respect to generalising over continuous action spaces. Although the lookup table assumption has resulted in further violation of the theory by practical considerations, at least Tesauro's TD-Gammon experiment suggests that we can still expect good performance providing appropriate generalisation techniques are used.

2.10 Generalisation techniques

In considering appropriate techniques for generalisation, the bridge has been crossed from the field of RL to the domain of statistics. Using a multi-layer perceptron to approximate the value function is only one approach to generalisation but any existing statistical technique for generalisation is a potential candidate for use on complex RL problems. Figure 2.3 illustrates the intended relationship.

An entire review of all generalisation techniques is clearly well beyond the scope of this thesis, but a representative sample will be considered shortly, as well as at various other places in the thesis. Therefore, to avoid duplication, only the briefest overview is given here and the reader is asked to accept a review of this subject as it is uncovered piece by piece according to the logical progression of the thesis.

2.10.1 Tiling the state space

Consider a robot learning problem in which there are two sensors each of which can take a value in the continuous range $[0, 1]$. The state space can then be represented as the unit square, and the task becomes one of dividing this space up into regions, with each region representing a discrete state of the standard RL problem. Deciding on the size, shape and number of these regions beforehand will generally be difficult, and the most obvious approach of using a high resolution grid superimposed over the entire space will result in an excessive number of states, particularly as the dimensionality of the space increases.

An alternative is to tile the space with overlapping regions, with each region corresponding to a hand-coded 'feature'. Then any particular state can be characterised by the set of features in which it appears. This particular approach is called *tile-coding* and is also referred to as the *Cerebellar Model Articulation Controller (CMAC)* because it models Cerebellar functionality (Albus, 1975). This is a specific instance of a broader set of basic generalisation techniques described as *coarse-coding*, in which state and action spaces are carved up in advance. A second instance of a coarse-coding algorithm is the *memory-based function approximator* where a continuous Q-function is represented as a set of prototypical Q-values. The Q-value of any point in the state-action space is approximated by a suitable combination of the surrounding prototype Q-values, scaled for example by the distance between each prototype and the point in the state-action space being evaluated. Updating the Q-value of an arbitrary point in the continuous state-action space can be achieved by altering the prototype values according to their relevance, and if there happen to be no prototypes suitably close to the current position then one can be spontaneously created. Coarse-coding approaches potentially suffer from the fact that generalisation is not adaptive. For example, in standard memory-based function approximators, the prototype positions are fixed. See

Santamaria et al. (1997) for a description and comparison of coarse-coding techniques.

Another simple approach to generalisation used by Mahadevan and Connell (1991) in their famous box pushing robot application is to represent each state explicitly (in their case the state space was discrete), but to update not only $Q(s,a)$ for the current state and action, but also $Q(s',a)$ for all $s' \in S$ within a fixed *Hamming distance* of s. This makes updating the Q-function over a large state-space more tractable. However, in this example, the only way each state could be represented explicitly was by performing hand-coded dimensionality reduction on the robot's vector of sensor readings.

2.10.2 Dynamic generalisation

A more appealing approach is to construct categories on-line, based on and in response to the input data. One approach, which is considered at length throughout this thesis, is to use Kohonen's Self-Organising Map (SOM) (Kohonen, 1987) to model the distribution of the input data. In the case of the two dimensional state space considered above, a SOM could be trained towards the two-dimensional sensory vectors generated by the robot. Each unit of the map would then be usable as a discrete state in the standard RL problem, with the SOM dynamically discretising the space with a variable resolution that reflects the robot's exposure to different parts of the environment. In work that is considered later in section 4.4, Sehad and Touzet (1994) and Touzet (1997) use a SOM to map the combined state-action-reward space. While the approach is favourably compared with a number of other representational techniques for a robot learning problem, the comparison is not detailed enough to conclude anything other than that the SOM is an interesting and potentially effective approach. The use of a Kohonen map to generalise over continuous state and action spaces is discussed and analysed at length during the course of this thesis.

In a similar vein, the *k-means* clustering technique (see Lloyd (1982)) could be used, or the *Adaptive Resonance Theory* network of Carpenter and Grossberg (1987b) which adopts a more constructive approach to the plasticity/stability tradeoff. Li and Svensson (1996) choose an ART network over a Kohonen network to represent the state space in a robot navigation problem. Their decision is based on the perceived inability of the latter to operate simultaneous learning and operation phases. However, results from chapter 5 suggest that this conclusion is invalid.

A dynamic approach adopted in Chapman and Kaelbling (1991) utilises *decision trees* in which an initially small number of states covering the entire space are iteratively split into smaller and smaller regions until each region behaves consistently with respect to the reward signal. This approach benefits from considering the reward information as well as the input data in generating categories, but it may not be suited to dynamic environments since the splitting is a one way process that could potentially result in too many regions being created.

In all these approaches, the emphasis is on dissecting the state space so that the value or action function can be maintained as a look-up table. Note that it is possible to perform the category construction and reinforcement learning processes in parallel, providing it is accepted that the categories underpinning the RL algorithm will change during learning. Tolerating moving states makes a further mess of the theory, but in practice such systems may still work well providing appropriate consideration is given to relative learning rates. This issue is considered in more detail in chapter 6.

2.10.3 Backpropagation

Other techniques directly approximate either the value function, or even the policy itself, thus side-stepping the need to maintain an explicit value function. Tesauro's TD-Gammon made effective use of the backpropagation algorithm to train the weights of a multi-layer perceptron (MLP) to perform powerful non-linear generalisation over the state-space. Similarly motivated is the *Complementary Reinforcement Backpropagation Algorithm (CRBP)* of Ackley and Littman (1990), in which a binary reward signal (intended to indicate whether the action is 'right' or 'wrong') is used to train a backpropagation network directly towards appropriate state-action pairs (see section 4.7. The approach is adopted by Ziemke (1996) in another robot learning problem to map robot sensors directly to optimally rewarded motor outputs. The scalability of CRBP is not clear, since the reward and outputs are binary. There is also no explicit interpretation of the action function which means the system operates as a black box.

The use of backpropagation is considered in more detail in chapter 8. Of particular interest is its relative robustness to the curse of dimensionality. For example, the input vectors of TD-Gammon had 198 dimensions!

2.10.4 Other techniques

A range of other techniques could also be conceived for generalising over the input space including the use of Radial Basis Networks (Powell, 1987), the GTM algorithm (Bishop et al., 1998), or an auto-associative MLP. A number of ad hoc techniques have also been suggested, such as the clustering algorithm of Mahadevan and Connell (1991) in which Q-values are grouped according to proximity in state-space *and* similarity with respect to the reward under each action. One interesting aspect of that work is that the state space is effectively decomposed differently for each action.

A complete comparison of these generalisation techniques with respect to the full range of reinforcement problems would be impossible. Different approaches will be more appropriate than others in different applications. A discussion of the applicability of some of these techniques with respect to specific problem features is given in chapters 6 and 8. In the meantime a number of relevant issues can be identified. For example, is generalisation adaptive or fixed? Is it linear or non-linear, supervised or unsupervised? Is the generalisation performed with respect to the reward function, the input data, both or neither. Also, a relevant question for non-stationary environments is how the tradeoff between plasticity and stability is addressed. A key distinction considered in chapter 8 is whether the representation of the state-space is *local*, as in the case of the Kohonen map for example, or *distributed* as in a backpropagation network. This may have implications when considering non-stationary environments, high dimensional state spaces, maintaining multiple actions, interpreting behaviour, and diagnosing faults. Other considerations include the resources required by the algorithm, the robustness and complexity of the algorithm, and potential for parallelisation, all of which may be particularly relevant in interactive domains such as robot learning. The consequences of pathological behaviour are also relevant. For example, what are the implications of getting stuck in a local minimum when using backpropagation, or producing twisted maps when using a SOM. All of these issues are encountered at various points throughout the thesis.

2.11 Summary

The theoretical foundations of reinforcement learning have been presented along with an account of the evolution of the most popular algorithms, including TD(λ) and Q-learning. Although in practice all of the assumptions required to guarantee convergence of Temporal Difference methods may be violated, performance is empirically found to be robust and good performance is usually achieved.

The problems of delayed rewards and continuous or prohibitively large state and action spaces are not failings of RL itself, but are inherent difficulties of the complex problems being addressed. The solution is not to say that RL is inadequate so let us invent something else, but instead to specifically address these issues, and indeed this is where much of the current research lies.

For any of the difficulties named above, or just because of an inadequate reinforcement signal, some problems may turn out to be too hard to solve using just the components of RL considered so far. In these more interesting cases a number of other ideas may need to be considered. As Kaelbling et al. (1996) note, almost all the advanced and interesting applications of RL utilise some form of innately specified knowledge about the task, be it cleverly crafted state representations (for example (Tesauro, 1994)), built-in behaviours (Maes and Brooks, 1990), reflexes (Li, 1999), pre-specified coordination of learning modules (Mahadevan and Connell, 1991), handcrafted progress estimators (Mataric, 1994, 1997), inbuilt heuristics for exploring actions (Wedel and Polani, 1996), and any number of assumptions about the nature of the reinforcement signal, or the environment model.

It is generally assumed that we are born with many innately specified tendencies, but that we also perfect our skills by increasing our knowledge of the world through trial and error interaction with our environment (Karmiloff-Smith, 1995). Looking to the animal world for inspiration yields other ideas including learning through imitation (see Hayes and Demiris (1994), and Price and Boutilier (2000) for RL-based imitation learning) and *shaping* (see Dorigo and Colombetti (1994) for an RL example) where an animal is trained incrementally on successively harder and harder versions of the task. With respect to domain specific learning, Gallistel et al. (1991) observe that animals have strong predispositions to learning certain kinds of associations. For example, a pigeon can be trained to peck a button for a food reward, but not to flap its wings. Conversely, it may be trained to flap its wings to avoid a shock, but not to peck. It is easy to see how biasing the kinds of associations that may be made in this way can greatly simplify the learning process. Even human development seems very constrained with certain things being learned at very specific times and being subject to specific constraints, prerequisites and processes (Karmiloff-Smith, 1995).

All this indicates that although we cannot expect to solve all problems by RL alone, the technique has still established itself as a valuable tool for our hardest endeavours. Designing RL-based solutions to interesting problems goes far beyond just picking one of the theoretical approaches plus a suitable set of parameters. It is the art of performing appropriate generalisation, maximising the information in the reward signal, and knowing how to bring all available information to bear on the problem, that transforms the theoretical and well understood foundations into a set of practical and exciting techniques.

CHAPTER 3

Neural Networks & Robot Learning

3.1 Introduction

Of particular interest to this thesis is the representation and generalisation of the action space in RL problems, and chapter 4 introduces a number of existing techniques for achieving this. However, first we introduce some preliminary material that underpins much of the thesis. Neural networks[1] feature particularly prominently, so a brief introduction to the field along with a description of some of the most relevant methods are given first. The second half of this chapter considers the behaviour based model of intelligence and the role and use of learning in robotics — a field which provides many practical problems of interest.

[1]There is sometimes confusion generated by the phrase *neural network* with respect to its ambiguity of reference to the real or artificial phenomena. The term is used in this account solely to refer to the artificial variety, unless otherwise stated, and is probably best paraphrased as *neurally inspired algorithms* or *artificial neural networks*.

3.2 Artificial neural networks

A number of examples of neural networks being used for representation and generalisation in reinforcement learning problems have already been encountered briefly, including the CRBP algorithm of Ackley and Littman (1990), the elevator scheduler of Crites and Barto (1996), and the *TD-Gammon* application of Tesauro (1994). The neural approach has turned out to be popular because of a number of properties of neural networks that lend themselves very generously to the RL cause. In particular, neural techniques have an incremental, iterative and interactive learning phase, embracing the philosophy that a complex problem can be solved by the iterative application of simple rules rather than the one-off application of a monolithic algorithm. These features are closely aligned with the principles of RL itself, making the two techniques naturally compatible. These features also have positive implications for robustness and tractability. Robustness is enhanced in the neural paradigm by distributed processing and representation. Also, an emphasis on sub-symbolic representation can, in the right circumstances, allow bottom-up solutions that remove from the designer the burden of providing appropriate symbols prior to learning.

But there are also difficulties associated with the neural paradigm including long training times, imprecision and lack of hard-constrainability (in the sense that there may be some post-processing necessary to convert a neural approximation into a legitimate solution), the need to find a suitable set of learning parameters, problems with local minima of error functions, and difficulty in interpreting solutions and diagnosing faults.

A full analysis of the advantages and disadvantages of neural techniques is beyond the scope of this thesis. However, we note that there are some good reasons for why neural networks have been, and will continue to be, a key technique for representation and generalisation in RL. The suitability of neural networks for RL applications is henceforth an assumption of the thesis.

Two particular models of interest are now considered: Backpropagation and Kohonen's Self-Organising Map (SOM).

3.2.1 The backpropagation model

The backpropagation algorithm [2] has been invented and re-invented numerous times in different domains, and from the simplest perspective of an application of the chain rule to non-linear regression problems, can be traced back to the numerical methods of the 1960s. Using networks of computing units was pioneered in the 1940s by McCulloch and Pitts (1947) although Warren McCulloch was considering networks of artificial neurons as early as 1927. The McCulloch-Pitts unit was later generalised by Rosenblatt (1958) to include adaptable weights yielding a simple learning machine known as the Perceptron. The perceptron was analysed in the 1960s by Minsky and Papert (1969) and its key failing of only being able to solve linearly-separable problems was identified. The disillusionment generated by this discovery was only overturned in 1985 with the advent of the now ubiquitous backpropagation algorithm (Rumelhart et al., 1986), which in principle is capable of achieving any continuous mapping to arbitrary accuracy. Backpropagation has since been applied successfully in a large number of domains to solving and generalising over large and difficult problems. Some celebrated examples include *NETtalk* (Sejnowski and Rosenberg, 1986, 1987) in which a network is trained to map an orthographic representation of the English language to a phonemic one, and more recently, the *TD-Gammon* application of Tesauro (1994).

The basic network architecture is shown in figure 3.1. The network consists of n *input nodes* or *units*, m *hidden* units, and p *output* units. Each connection between two units has a *weight*, with the weight between input unit i and hidden unit j being represented by the term w_{ij}, and the weight between hidden unit j and output unit k represented by the term w_{jk}. Each hidden and output node computes an *activation function* that depends on the weighted sum of its inputs. If the activation of a unit is given by $\xi(unit)$, then:

[2]Backpropagation is a learning algorithm applied to the weights of a feedforward (or recurrent) network of computing units. However, because it is so common to see the two used together, this thesis often uses the word *backpropagation* figuratively to refer to either the network type (as opposed to the SOM for example) or the entire algorithm. It is assumed that context alone will disambiguate its reference.

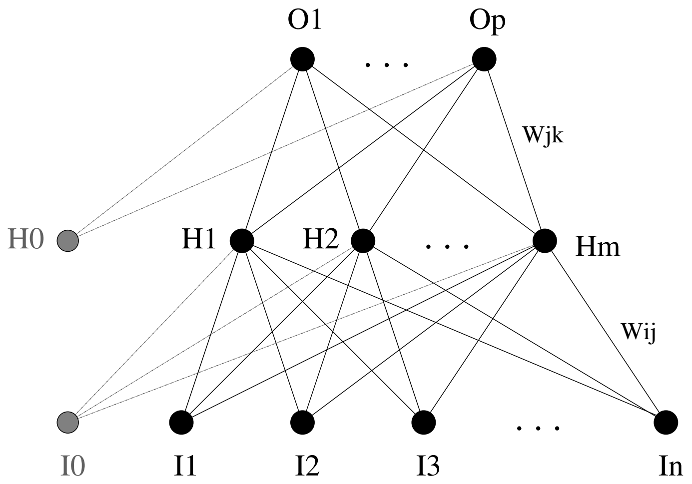


Figure 3.1: A network of units for training with the backpropagation algorithm. Each node computes a differentiable function of the weighted sum of its inputs. Each unit also maintains a *bias* which can be thought of as an extra weight on a connection from an extra unit with a constant activation of 1. This interpretation of the bias is shown on the diagram in grey. With enough hidden units and appropriate selection of the weights on the connections between units, any continuous function from the inputs to the outputs can in principle be approximated to arbitrary accuracy.

$$\xi(H_j) = f\Big(\sum_{i=0}^{n} w_{ij}\xi(I_i)\Big) \quad \text{and} \quad \xi(O_k) = g\Big(\sum_{j=0}^{m} w_{jk}\xi(H_j)\Big) \tag{3.1}$$

where f and g are differentiable functions which may, but need not, be the same. In actual fact, every node could in principle have its own activation function, although it is common to use the standard *sigmoid* function of figure 3.2 for all units. The input units are assumed to have an activation corresponding to the input data at that node. Each hidden and output unit also has a *bias* which is added into the weighted sum for that unit, and can be treated as an extra weight from a unit which always has an activation of 1 (these units are shown as $I0$ and $H0$ in figure 3.1).[3]

Given a large enough layer of hidden units, and an appropriate set of weights, it can

[3]The purpose of the bias in the perceptron is to translate the decision boundary away from the origin. For example, without biases, an input vector of all zeros is obliged to produce an output vector with every element $= g(0)$. The same principle underpins the bias in a backpropagation network — it translates the mean activation of the output units to the mean activation of the target distribution (Bishop, 1995). The functional variation is then provided by the other weights.

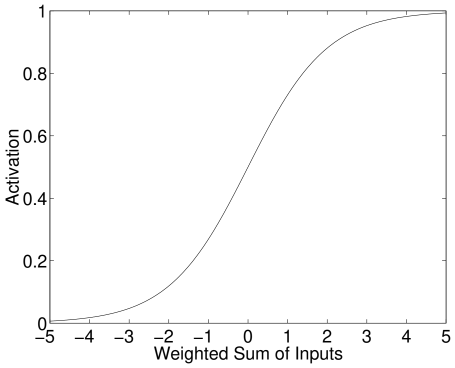


Figure 3.2: The standard sigmoid activation function: $y = \frac{1}{1+e^{-x}}$. It is chosen for convenience of differentiation and its approximation to the original step function of the McCulloch-Pitts unit. Other functions are also used such as linear functions (usually in the output units), and the *tanh* function which is often empirically found to result in faster convergence (Bishop, 1995, pg 127).

be shown that any continuous function from n inputs to p outputs can be approximated to arbitrary accuracy (see Bishop (1995) (pg 130) for an outline proof based on Jones (1990) and Blum and Li (1991)). However, no practical constructive proof exists and so actually finding a suitable set of weights is a challenge.

The learning problem consists of a set of training samples that comprise T input-output vector pairs $< \mathbf{x}^t, \mathbf{y}^t >$ with $t = 1 \ldots T$. The aim is to find a set of weights which allow the network to accurately map each input vector to the corresponding output vector. To this end, the backpropagation algorithm first defines the *Error* of a particular output unit as the square of the difference between the *actual* output of that unit for the current input vector given the current weights of the network, and the *target* output for the current input according to the training data. Other error measures may be used, but with this particular approach the output of the network can be shown to converge on the *expected* target for that input[4]. An expression for the overall error of the whole network given the current weights ($\mathbf{W}$) can then be given by summing over each training sample and each output of the network:

[4] Bearing in mind that the training data may be drawn from a target distribution with non-zero variance.

$$Error = \frac{1}{2}\sum_{t=1}^{T}\sum_{k=1}^{p}\left(y_k^t - \xi\big(O_k(\mathbf{x}^t|\mathbf{W})\big)\right)^2 \tag{3.2}$$

where y_k^t is the *target* activation of output unit O_k for input $\mathbf{x}^t$, and $\xi\big(O_k(\mathbf{x}^t|\mathbf{W})\big)$ is the *actual* activation of output unit O_k for input $\mathbf{x}^t$ given the current set of network parameters, $\mathbf{W}$.

This error measure can then be minimised by iterating the following procedure:

❶ Select a training pair at random.

❷ Generate the actual network output for the input of that pair.

❸ Calculate the Error, E, of the network for this particular pair using (3.2) (without the $\sum_{t=1}^{T}$ sum).

❹ Calculate $\frac{\delta E}{\delta weight}$ for each weight of the network.

❺ Update each weight negatively proportionally to this gradient.

In practice, $\frac{\delta E}{\delta weight}$ is easily calculated for each weight of the network by using partial differentiation to propagate an appropriate portion of the Error term back through the network to the weights according to their relative responsibility[5]. Assuming that an appropriate learning rate is used, then by updating the weights so as to reduce the Error value for each training sample in turn, a minimum of that Error function is bound to be found for the entire training set. However, there is no guarantee that this minimum is the lowest point on the Error function because it may be one of the typically many *local minima* that is found.

The implications of a number of properties of the backpropagation algorithm for reinforcement learning are considered in more detail in chapter 8. For now some key properties are defined as:

[5]See Bechtel and Abrahamsen (1991) (pp.88-91) for a compact derivation of the backpropagation algorithm. For a more elegant, novel and graphical illustration see Rojas (1996) (chapter 7).

- The ability of a trained network to generalise from training samples to unseen inputs is powerful (as seen in both *NETtalk* and *TD-Gammon*). The generalisation is non-linear in the sense that the network outputs are a function of two layers of weights, with the second layer of weights applied to the outputs of the first layer.

- The algorithm scales well with the dimensionality of the problem (*TD-Gammon* had nearly 200 input dimensions).

- Key problems are convergence to local minimum, and prohibitively long training times (*NETtalk* took 12 hours to train. *TD-Gammon* took weeks). Other problems include sensitivity to parameters and initial conditions.

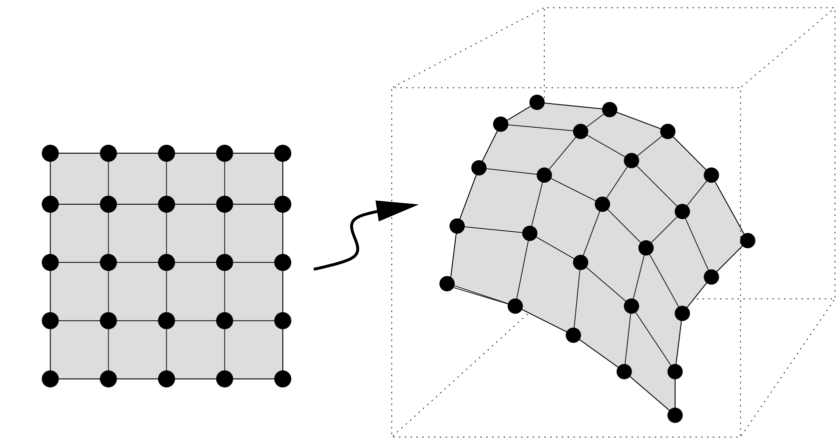
- While more sophisticated algorithms than standard backpropagation exist (see section 8.4.1), potential problems with local minima and long training times persist.

- The representation is *distributed*. Every output depends to some degree on every weight and every input. Similarly, weight changes are made globally across the entire network during each update. This adds to the robustness and generalising power of the network, but may be a disadvantage if we desire stability in one part of the function while attempting to adapt another part. An example of this is encountered in section 8.5.3.

For an authoritative exposition of the principles underpinning backpropagation, see (Bishop, 1995; Rojas, 1996).

3.2.2 The Kohonen map

A second popular neural algorithm is Kohonen's Self-Organising Map (SOM) (Kohonen, 1987, 1995). In isolation the SOM is an example of unsupervised learning since only inputs are provided. Although this is in contrast to backpropagation, which is supervised by explicit provision of input-output pairs, both algorithms turn out to be highly applicable to representation and generalisation in RL.



Kohonen Map in Physical Space   Kohonen Map in Weight/Input Space

Figure 3.3: A two dimensional Kohonen map is shown in *physical space* on the left and in *weight space* (or *input space*) on the right. The input data lies on a two-dimensional manifold despite the input space being three-dimensional.

In its standard form, the Kohonen map consists of a two-dimensional grid of units, with each unit being identified by its position within this grid. Associated with each unit, t, is a weight vector, $\mathbf{w}^t = [w_1^t, w_2^t, \ldots, w_D^t]$ where D is the dimensionality of the input data. The learning problem this time is to find a suitable set of weights for each unit so that the network models the distribution of the input data in the input space (also the weight space). In many cases, the *intrinsic* dimensionality of the distribution may be low, even though the dimensionality of the input data itself is large. In these cases the SOM can be used as a dimensionality reduction technique. As an illustration, consider figure 3.3 in which a map is shown in physical space on the left, and in weight (input) space on the right. In this example, the two-dimensional Kohonen map is an appropriate choice for representing the three dimensional data because the data happens to lie on a two dimensional surface inside the input space.

The learning rule responsible for finding a suitable set of weights is simple: Given an input vector, $\mathbf{x} = [x_1, x_2, \ldots, x_D]$, the distance between each unit, t, of the Kohonen map and that vector is calculated by:

$$\text{Manhattan Distance} = \sum_{d=1}^{D} |x_d - w_d^t| \ \textbf{or} \ \text{Euclidean Distance} = \sum_{d=1}^{D} \left(x_d - w_d^t\right)^2 \quad (3.3)$$

The unit with the smallest distance is that which most closely represents the current input, and is thus considered the *winner* for that input. The weights of the winning unit are now updated towards that input:

$$\mathbf{w}^{winner} = \mathbf{w}^{winner} + \alpha(\mathbf{x} - \mathbf{w}^{winner}) \quad (3.4)$$

where α is the learning rate. Now the map more faithfully represents that input vector. The process is iterated for each input vector in the data set effectively resulting in a competition between different regions of the input space for units of the map. Dense regions of the input space will tend to attract more units than sparse ones, with the distribution of units in weight space reflecting the distribution of the input data in the input space.

Topology preservation

An important feature of the SOM is that in addition to the winning unit being updated towards the current input vector, the winning unit's *neighbours* are moved in this direction too. The definition of a neighbour is a unit which is close to the winner in the physical (or topological) space of the map. A *neighbourhood function*, $\psi(winner, t)$, is defined that weights the update of unit t towards the current input vector according to its distance from the winning unit in physical space. So the full learning rule for an input, $\mathbf{x}$, and winning unit, *winner*, is:

$$\mathbf{w}^t = \mathbf{w}^t + \alpha\psi(winner, t)(\mathbf{x} - \mathbf{w}^t) \quad (3.5)$$

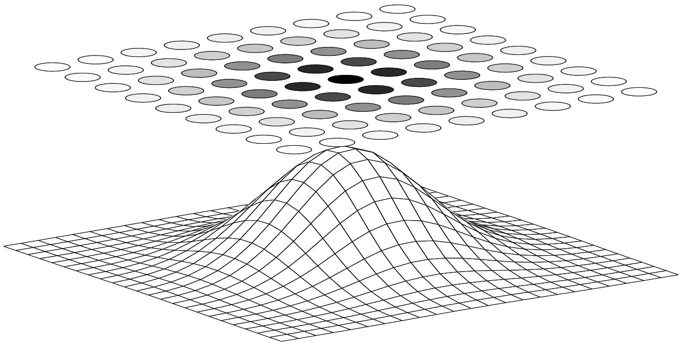


Figure 3.4: A common neighbourhood function in which the influence of the winning unit on near neighbours is greater than on those at a distance. The strength of shading of the units in the map reflects the height of the neighbourhood function underneath, and corresponds to the value of the term $\psi(winner, t)$. In this case, the winning unit happens to be in the centre of the map.

for each unit, t, of the map. Updating entire neighbourhoods rather than single units helps ensure that the distribution of the map reflects the distribution of the data, and also yields the famous *topology preserving* property of the SOM. Because units effectively learn from the experience of their neighbours, they tend to end up next to each other in weight space as well as in physical space, with the effect that the topology of the input data is preserved in the trained map. A typical neighbourhood function is shown in figure 3.4. Both the neighbourhood function and the learning rate are usually reduced throughout the learning process so that initial quick but coarse judgements are slowly refined to yield a stable mapping that models the input distribution.

As an explicit illustration of the Kohonen map at work, consider figure 3.5 which shows a two dimensional SOM mapping a uniformly sampled two-dimensional input space. The SOM contains 10×10 units and uses a neighbourhood function which decreases linearly with the distance from the winner. The neighbourhood is annealed from an initial size of 7×7 units to 1×1 as learning progresses. The learning rate is also annealed from 0.3 to 0 during learning. Figure 3.6 shows the flexibility of using one and two dimensional maps on a number of other different distributions. In principle a Kohonen map of any dimension may be used, but note that using a map with a dimensionality less than the intrinsic dimensionality of the data (as in (b),(d),(f) and (g)), results in compromising topological preservation. In practice, two dimensional maps are usu-

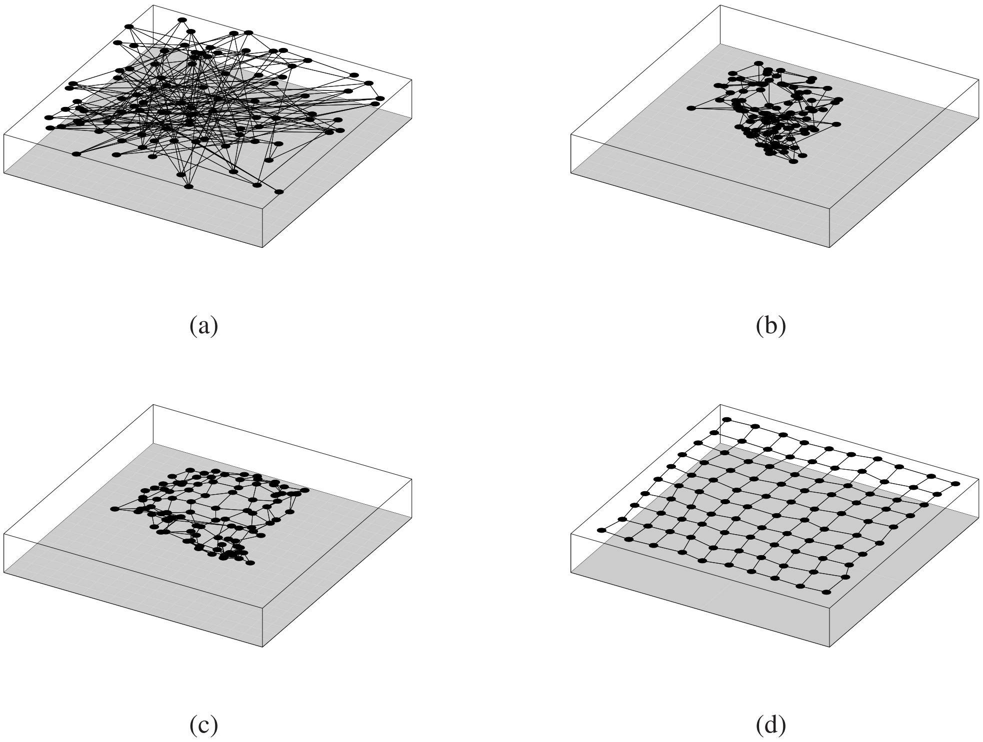
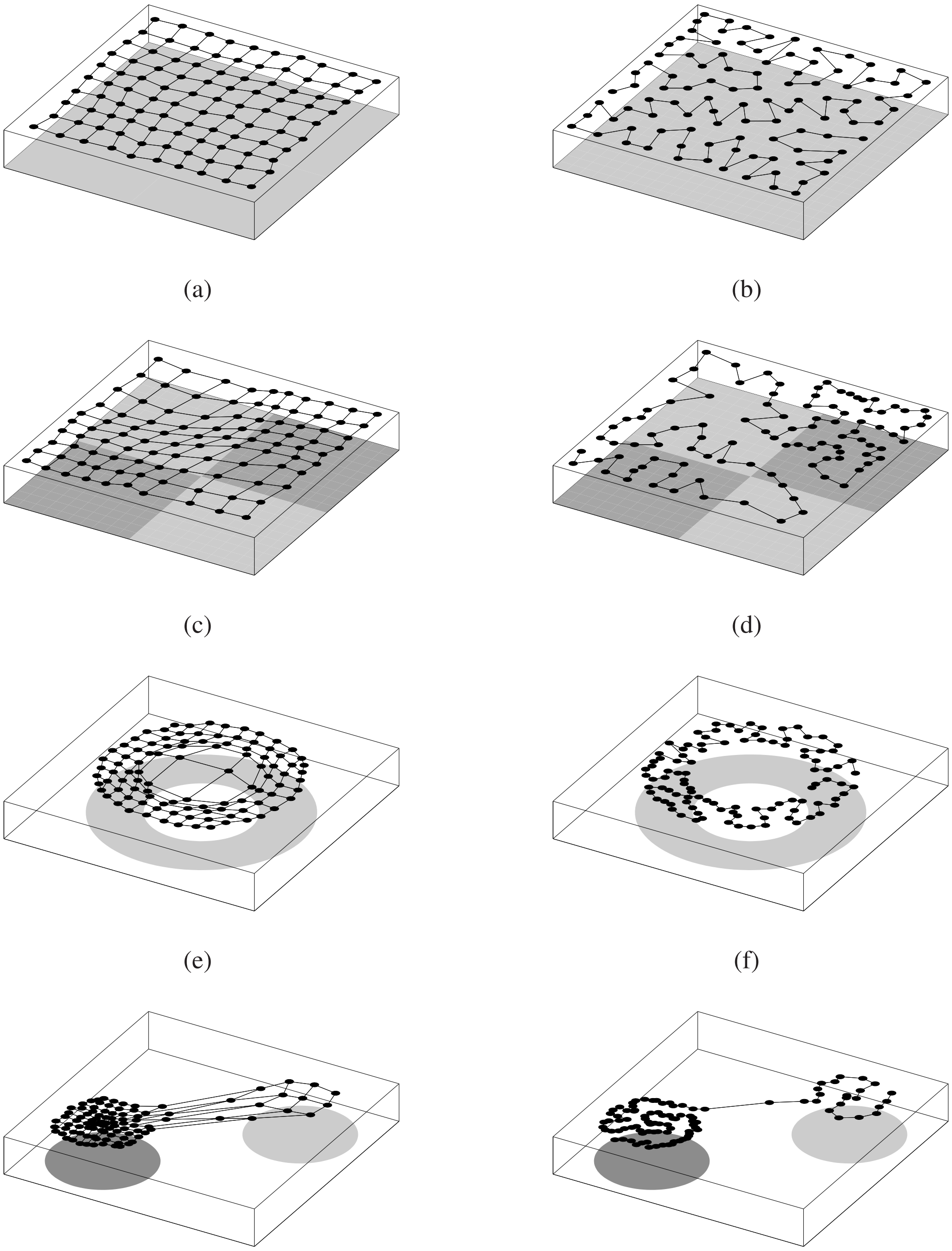


Figure 3.5: The Kohonen network plotted in the weight/input space at various points during formation. In figure (a) the initially random weight vectors mean that the units occupy random positions in the input space and there is no correspondence between neighbours in physical space and neighbours in weight space. As the map forms through figures (b) and (c), the units begin to distribute themselves equally over the space and neighbouring units begin to occupy neighbouring points in weight space. Finally figure (d) shows the final state of the map with faithful representation of the equally-distributed data, and complete topological preservation.

ally favoured, partly because of the straight forward implementation and visualisation, and partly because higher dimensional maps will tend to require a prohibitively large number of units.

As with backpropagation, the SOM is analysed in greater detail in following chapters, so only a summary of the main issues surrounding the algorithm is given here.

- The distribution of units in a trained map will reflect the distribution of the input data.



(a) (b)

(c) (d)

(e) (f)

(g) (h)

Figure 3.6: The figures on the left show a two dimensional Kohonen mapping of various two dimensional spaces. The shading underneath each map reflects the actual density of the input distribution (dark grey = 4 × light grey): (a) uniform contiguous, (c) non-uniform contiguous, (e) irregular, and (g) discontiguous, non-uniform and irregular. The figures on the right show the mapping of the same distributions obtained using a one-dimensional map.

- Topology is preserved so that similar inputs generate activity in similar regions of the map.
- A representation that reflects the intrinsic dimensionality of the data can be achieved.
- The Kohonen map is not as powerful a generalisation tool as backpropagation. The data is represented explicitly, with the number of units (and therefore the number of free parameters) scaling linearly with the size of the input space (or exponentially in the number of dimensions). This is in contrast with backpropagation in which the number of hidden units required grows much more slowly with the size of the problem.
- An essentially local update rule allows learning in one part of the network to take place without disturbing the remainder of the representation. This is a double-edged sword which is investigated further in chapter 8.
- Some advantages of the SOM are that it is simple and easy to implement with a short processing cycle. It is intuitively appealing with easy visualisation, interpretation and diagnosis, and it is widely used so a large corpus of empirical data exists.
- Some disadvantages include a lack of convergence proof for anything but the one-dimensional case and a lack of an energy function to minimise making principled analysis difficult. Finding appropriate learning rates, network dimensionality and neighbourhood functions is an empirical task incumbent on the designer, and it is also known that in general the distribution of the input data is not faithfully represented by the network, with the map tending to under-sample high probability regions and over-sample low probability regions (Hertz et al., 1994).

Biological parallels

The SOM has its roots in biology, although the exchange of non-local information in selecting the winner is a minor computational drawback to parallelisation and biological plausibility. Experiments on cats blinded in one eye during development have shown that the visual cortex is able to adapt so that that part which formerly responded to the blind eye is re-mapped to respond to the good eye (Edelman and Finkel, 1990; Edelman, 1987) in a manner not dissimilar to the operation of the Kohonen map. Of particular biological relevance is the topology preservation found in the mammalian somatosensory and motor cortices in which neighbouring body parts are represented next to each other (Rojas, 1996, pg 392). Topological mapping has been further implicated in the mammalian nervous system by the similarity between ocular dominance patterns found in the visual cortex and results of simple experiments performed with two-dimensional Kohonen networks trained to map a three dimensional space (Goodhill (1992) for example).

The Kohonen map is one of the most widely used unsupervised learning methods and has been proposed for a huge number of applications including the Travelling Salesman Problem (Angeniol et al. (1988). See also Durbin and Willshaw (1987) for the related *elastic net* algorithm), the Cart-Pole Balancing problem (Ritter and Schulten, 1987) and robot arm coordination (Rojas, 1996). For a list of thousands of papers based on the Kohonen map, see Kangas and Kaski (1998), and for a book of varied applications and analyses see Oja and Kaski (1999).

3.3 Learning in robots

Since some of the experimentation and discussion contributing towards this thesis addresses the problem of robot learning, a brief overview of this subject is now offered. The justification for studying learning within a context of autonomous robots was given in chapter 1. In short, by considering agents which physically interact with the real world[6], a number of issues which pertain to *any* effective and scalable algorithm are automatically addressed. Such issues include timeliness of response, process efficiency, algorithmic compactness, robustness to noise and incomplete data that arrives at unpredictable times, the need for effective generalisation, an ability to handle non-stationary environments, as well as general scaling issues such as parallelisability and complexity.

[6] Or in the case of this thesis, a simulated world.

A second reason for being interested in robotics is that from a philosophical perspective this thesis is largely motivated by the goal to build intelligent, adaptable systems. To this end, robots may be seen as a well-defined experimental platform, RL as a suitable design paradigm, and neural networks as an appropriate implementational paradigm.

A third interest in robotics can be phrased as a challenge, but this is postponed until the end of the section.

3.3.1 The behaviour based model of intelligence

Consider the difference between a termite mound and the Empire State building. During the construction of the man made article, it would have been possible to enter the architect's office and view the plans for the building at various levels of abstraction. The plans would have had a hierarchical structure with sketches of the whole building at the top, floor plans somewhere in the middle, and individual structural components such as doors and windows somewhere near the bottom. It would have been possible to enter offices and see a similar hierarchy of human organisation with executives at the top, managers in the middle, and workers at the bottom. But look inside a termite colony and there is no explicit plan of a termite mound, either on paper, or likely in any individual termite's mind. Neither is there any sign of a complex hierarchical management system beyond the very broadest division of labour.

In the mid nineteen-eighties, researchers led by Brooks, Minsky and Braitenberg[7] pioneered a change in emphasis in the approach of the artificial intelligence community to the design and construction of its artefacts. The shift was away from the Empire State model and towards that of the termites. The traditional approach largely involved breaking the problem down into the sequential phases of receiving input, building a model of the world from that input, constructing a plan based on that model, and then executing the plan. The basic idea is shown in figure 3.7 and is often characterised by a large amount of symbolic, hierarchical processing in the modelling and planning

[7] It is not clear who inspired who, but it appears to the author that ideas found in 'Vehicles' (Braitenberg, 1984) and 'The Society of Mind' (Minsky, 1986) may have laid the foundations for Brooks' Subsumption Architecture (Brooks, 1986). However, since the three were published within a few years of each other, the exact origin of these ideas remains unclear.

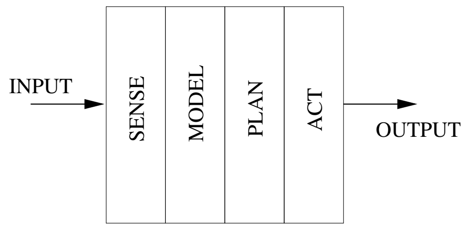


Figure 3.7: A broad characterisation of *traditional AI*

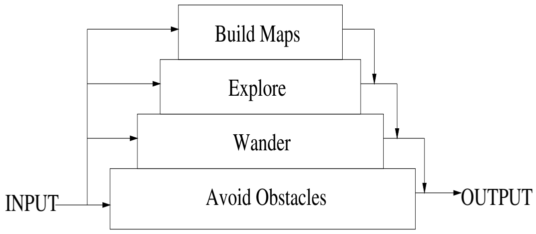


Figure 3.8: An alternative, *Behaviour Based* 'subsumption' approach

stages. An explicit alternative pioneered by Brooks (1986) is illustrated in figure 3.8 (adapted from Brooks (1986)), where the control system can be thought of as being decomposed into parallel task achieving behaviours rather than into the processing elements of sequential pipeline. The idea is that each *layer* or *behaviour* is incrementally built on top of simpler lower layers, with lower level behaviours having priority but with higher level behaviours able to take control as and when they are able to suggest something sensible. For example, in the figure, the relatively complex process of map building is based on a foundation of fundamental supporting behaviours which are dedicated to coping with the ever imminent need to survive in and move safely around an unpredictable and dynamic world. The main directive is that each behaviour is simple and reactive, involving no modelling or planning with the intention that complex behaviour will *emerge* from the interaction of simple behaviours with each other and with a complex environment.

The behaviour based approach can be paraphrased more generally by figure 3.9. The system is divided into parallel, competing behaviours, each performing a minimum amount of modelling and planning. Each behaviour typically receives domain specific input and performs domain specific computation, resulting in a kind of committee of experts. The usual difficulty of such systems is deciding how to integrate these multiple behaviours so that the appropriate behaviour fires at the appropriate time, and so that the desired global behaviour emerges.

One solution is offered in the box pushing robot of Mahadevan and Connell (1991). The aim of the robot was to move around a room finding and then pushing boxes to

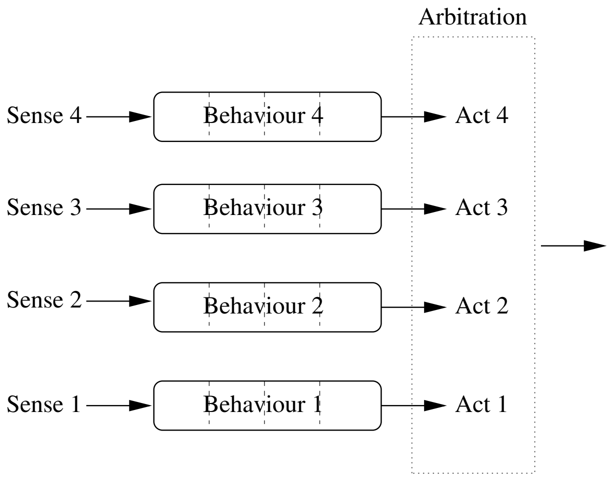


Figure 3.9: A more general behaviour based scheme. In a single agent system, some form of arbitration is usually required between the parallel outputs if they are competing for the same resources. Outputs may overwrite each other, combine with each other, or even feed into other behaviours as inputs. The model is also broadly applicable to multi-agent systems. For example, in the termite colony example, each termite may be represented as a distinct behaviour, with concurrent actions being possible.

the walls. The authors employed three behaviours constructed as in figure 3.10. A *finder* behaviour is responsible for locating boxes and is always active unless a *pusher* behaviour detects that the robot is already in contact with a box. As long as this is the case, the pusher behaviour overrides the finder unless the *unwedger* detects that the box being pushed is stuck against a wall, in which case the unwedger assumes command and frees the robot to look for another box. Hence the behaviours are prioritised with the unwedger given ultimate precedence, and the finder assuming the role of a default behaviour. This approach of an overriding priority hierarchy has an intuitive appeal for resolving the conflicts generated by multiple behaviours, and is commonplace in the literature (see also (Shackleton and Gini, 1997; Kube and Zhang, 1994) for example).

One could easily conceive of the termite colony being constructed in a similar way, with each termite representing a behaviour. Moreover, each termite might have its own set of internal behaviours which are triggered by the environment. Now, through years of evolution of this behaviour based system, a society of termites may produce the large and complex structure of a termite mound, even though there is no central coordinator, no seat of intelligence, and no explicit maps, plans or models of either the world or the task in hand.

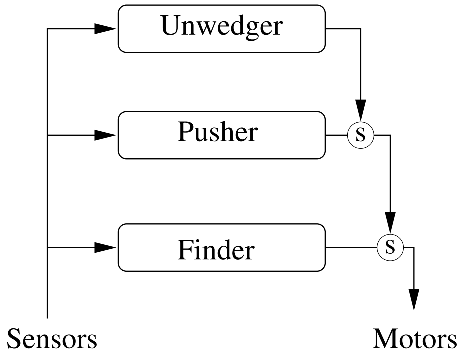


Figure 3.10: The behaviour layering approach used by Mahadevan and Connell (1991). The nodes labelled with an *s* indicate that the upper behaviours can override or *suppress* the behaviours in the lower layers.

Around this time, a series of robots performing a number of real-world tasks including wandering around an office avoiding collisions with obstacles and people were built (see Brooks (1986, 1991b))[8] . The systems, based on Brooks' *subsumption architecture* of figure 3.8, were found to exhibit robust and timely behaviour which could mistakenly appear to the observer to be the product of a centrally coordinated system performing planning and deliberation. The *reactivity* of each behaviour and the system as a whole vastly improved robustness since it was no longer necessary to perform extended symbol manipulation to arrive at a course of action — a process which is vulnerable to the use of inappropriate symbols, and which may take a significant amount of time during which the environment is likely to have changed anyway. These observations led Brooks to suggest that "the world is its own best model" (Brooks, 1991a). Now looking inside the robot for a 'seat of intelligence' is as worthless as looking inside each termite of a colony for an architectural plan of the mound. The notion of complexity and indeed intelligence itself had been removed from a set of symbols inside the agent, and was now considered instead to be found in the environment itself and the observer's mind[9] [10] [11].

[8]Tasks which turn out to be harder to achieve than one might imagine.

[9]Brooks used phrases like *"Intelligence is determined by the dynamics of interaction with the world"* and *"Intelligence is in the eye of the observer"* (Brooks, 1991a).

[10]The significance of the environment in producing the desired behaviour explains why behaviour based researchers are such strong advocates of using real physical robots rather than simulations in which important environmental features may be lost.

[11]Brooks' subsumption architecture is an instance of the behaviour based model in which specific rules are provided for how behaviours can communicate and interact with each other. In an excellent

Minsky (1986) provides the appealing analogy of a box which contains a mouse. Looking for the property of “can contain a mouse” in any side of the box or any individual nail that holds those sides together is fruitless. Similarly hopeless is trying to decompose the property and looking for its component parts in the components of the box. Only when each piece is considered in interaction with all the other pieces does the property of containment emerge. Gestalt properties of this kind where the whole is the sum of the parts *plus* their interactions with their environment and the observer, characterise “emergence”.

While natural systems of this kind are ubiquitous, there ought to be a good reason for why the Empire State approach was, and usually still is, favoured in human endeavours. A key difficulty of the behaviour based approach lies in translating a desired global behaviour into a suitable set of behaviours that are composed in an appropriate way. Indeed, the best that behaviour based models of AI have been able to produce so far falls very short of even insect-like intelligence, and comes nowhere close to realising those original aspirations of traditional AI researchers of understanding and modelling vision, logic, language, and other ostensible properties of human intelligence. There is still much debate as to how the behaviour based approach can be scaled to re-excite the interest of traditional AI (Arkin, 1991, pg 28). Tsotsos (1995) argues that “the strict behaviourist position for the modelling of intelligence does not scale to human-like problems and performance”, whereas Brooks (1990) maintains that “...in principle we have uncovered the fundamental foundation of intelligence”. Although the traditionalist and behaviour based camps often appear to advocate mutual exclusivity, the two may yet learn to cohabit for their mutual benefit. This opinion is reflected in Arkin (1991): “The false dichotomy that exists between hierarchical control and reactive systems should be dropped”, and is vindicated by the current trend towards precisely this kind of coalition.

In the meantime, if the traditionalists become frustrated with the inability of the behaviour based paradigm to scale beyond basic insect-like intelligence, Brooks (1991b) reminds us that intelligence (at least in terms of the traditionalist’s goals) has only ap-

paper, Prescott et al. (1999) consider the biological plausibility and motivation of the subsumption architecture, and by drawing on a range of neurophysiological data and ethological experiments succeed in establishing clear parallels between Brooks’ approach and animal brains.

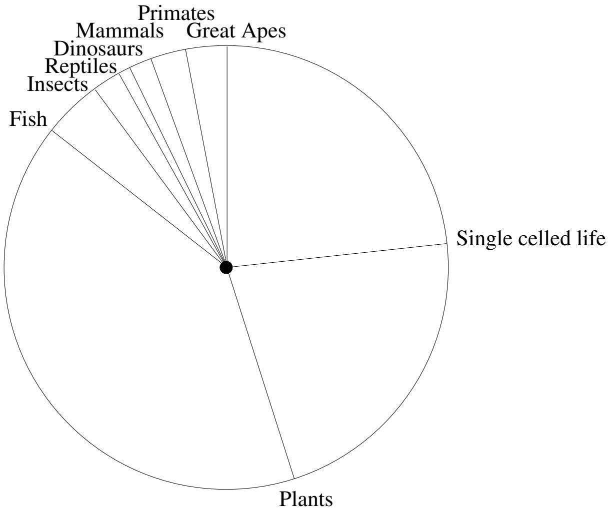


Figure 3.11: The evolution of intelligence, following Brooks (1991b).

peared very recently on our planet. In fact, if the history of earth is represented as an hour on a clock face as in figure 3.11, single celled life did not appear until almost a quarter past the hour. Plants made their debut at half past and the first fish at just eight minutes to. Insect, reptiles and dinosaurs were quick to follow, but mammals did not appear until three minutes to the present day. The first primates came into existence ninety seconds ago, and man himself just three seconds ago. Agriculture was invented around one hundredth of a second ago and computers just half a millionth of one second ago. One possible interpretation of this data is that intelligence pertaining to reacting, moving and surviving in a complex, dynamic and unreliable world represents a necessary foundation for higher order intelligence, with apparently abstract competences such as logic and language representing only the smallest visible tip of the iceberg.

3.3.2 Learning and behaviour based intelligence

One of the key constraints advocated by pioneers of the behaviour based paradigm was that a minimum amount of representation should be used within each behaviour in order to maximise reactivity and therefore also the agent's ability to respond to changes and complexity in the environment. Nevertheless, scaling the purist approach has proved to be difficult, as recognised by Brooks himself:

> "As the number of behaviours goes up, the problem of control and coordination becomes increasingly complex. Additionally, it is often too difficult for the programmer to fully grasp the peculiarities of the task and environment, so as to be able to specify what will make the robot successfully achieve the task." (Maes and Brooks, 1990).

To address this, much effort has been expended in attempts to ascertain how hybrid architectures may be built out of both reactive behaviour based components *and* more traditional hierarchical, top-down approaches (see Connell (1992), Lyons and Hendriks (1995), Arkin (1986), and the Procedural Reasoning System of Georgeff and Lansky (1987)). As a proof of concept, Arkin (1991) draws on psychological evidence from Shiffrin and Schneider (1977) for the existence of two distinct modes of behaviour in the brain — willed and planned. A striking illustration of this distinction is also found in sufferers of Parkinson's disease who, as a result of cell death in the basal ganglia, are unable to perform conscious willed actions, even though the ability to produce unconscious reactions such as catching a ball thrown at the patient may be left intact (Gillies, 2001). This suggests that there may be some distinction in the brain between reactive and deliberative behaviour that, ordinarily at least, we are not aware of.

So the key question is how can learning, modelling and planning be used to enhance the behaviour based stance. Of particular interest to this thesis is the issue of learning and in particular how reinforcement learning and the behaviour based approach may be combined. Mahadevan and Connell (1991) used RL to learn each of the three behaviours of figure 3.10, while Maes and Brooks (1990) fix the behaviours but use an ad-hoc RL technique to learn the coordination between those behaviours. Other approaches have attempted to learn both behaviour *and* coordination (Dorigo and Colombetti (1994) and Singh (1992) for example). These and other experiments suggest that RL and the behaviour based approach may indeed be used to complement each other. However, while both have been used independently to break new ground (*TD-Gammon* of Tesauro (1994) and the robust office robots of Brooks (1986, 1991b) respectively), it appears that there are still no examples of the two being used together to produce a ground-breaking application.

This raises the question of whether learning in behaviour based systems and indeed robotics in general is productive or not. The argument in favour is apparently eroded further by looking towards the natural world for inspiration. Nature appears to use learning as something of a last resort. Insects seem to have most of their behaviour hardwired with only a few exceptions such as the way in which bees can memorise food sources or promising locations for new hives (Dawkins, 1982)[pg. 205]. Other animals seem to exhibit very constrained and special purpose learning (Gallistel et al., 1991). Examples that we encountered earlier come from experiments on pigeons which are shown to be readily able to learn to peck for a food reward, but not to avoid a shock. Conversely, it has been found that they can be trained to flap their wings to avoid a shock, but not to receive food. Similarly, hamsters can learn quickly to dig, scrabble or rear for a food reward, but slowly or not at all to scratch or scent mark (see Gallistel et al. (1991)). Clearly there is an advantage in constraining learning to useful associations since learning is costly in terms of both the time spent doing it, and the potential risks of exploration.

Given that all behaviour has to be learned sometime, whether it be in "evolutionary time" (Bryson, 1996) or the lifetime of an agent, then a natural question is: which is better? Looking to nature for inspiration suggests that the experience of evolution may often be preferred over that of the animal, except in those circumstances where the environment changes quicker than evolution can adapt (such as a bee's source of food). That hardwired behaviour may be sufficient for such a large proportion of animal behaviour is a testament to the robustness of those behaviours, and this in turn is likely to be a consequence of the reactivity of these behaviours. So the argument may go something like this: "Nature seems to prefer to hardwire behaviour when it can so as to avoid the costly and potentially dangerous process of lifetime learning. This seems to imply that hardwired behaviour can be highly successful if designed appropriately (as in the case of the termites), and secondly that lifetime learning is difficult given that the animal itself will likely have less experience than the evolutionary process itself."

However, making intelligent robots turns out to be very different from making animals for two important reasons. The first is that because of the implications of natural selection, the main goal of any animal becomes survival. Intelligent behaviour, however we choose to interpret it, may aid survival but is by no means evolution's mandate. With intelligent robots, the opposite is true. The primary goal of AI is intelligent behaviour, and survivability is less important. For example, a robot could be sent to learn how to achieve a task in an unknown environment. If it failed to learn or even to return then another one could be sent out, and so on until one managed to return having acquired suitable behaviour. This behaviour could then be extracted and hardwired into subsequent machines. In short, notwithstanding economic considerations, the cost of lifetime trial and error may be more acceptable to a robot designer than an animal.

The other key difference is that evolution currently has far more experience of making animals than engineers do of robots. The former can run trillions of parallel experiments over billions of years, a luxury clearly not yet available to robot designers.

The conclusion is that even if nature does favour hardwiring over lifetime adaption, the mandate, experience and circumstances of the robot designer are sufficiently different to warrant scepticism. Scaling the behaviour based model of intelligence may yet benefit from learning, but no conclusive evidence one way or the other has yet been provided. This is the challenge alluded to at the beginning of the section. Tesauro's *TD-Gammon* was a fine example of a ground breaking application of backpropagation and RL, in that the performance of the resulting system was unprecedented. The same can be said of the early behaviour based robots (see Brooks (1990)). However, as yet RL and the behaviour based approach have not been combined to achieve the same kind of ground breaking accolade, and this remains a thrust of current research. Although this thesis is primarily concerned with RL, and not behaviour based robotics, a new model for the reinforcement learning of real-valued actions proposed in chapter 5 will be applied to a simple behaviour based system as part of this ongoing research. Indeed, this thesis started out with the intention to investigate suitable learning techniques specifically for behaviour based models of intelligence. The legacy of this intention is found in section 5.6.5, but is also apparent at various other points in the thesis where the subject of learning robots is used to motivate, inspire, and contextualise the discussion.

	NN	RL	BB
Incremental	✓	✓	✓
Interactive	✓	✓	✓
Iterative	✓	✓	✓
Asynchronous	✓	✓	✓
Simple constituents - emphasis on Emergence	✓		✓
Distributed representation	✓		✓
Distributed processing	✓		✓
Biological parallels	✓	✓	✓

Figure 3.12: Some common features of different approaches to intelligent, adaptive behaviour.

3.4 Summary

To conclude this chapter, we briefly summarise in figure 3.12 some of the key computational features of RL in comparison with both the behaviour based (BB) model of intelligence, and neural networks. In pursuit of the goal of intelligent, adaptive behaviour, a key challenge is likely to be finding the correct places at the correct levels for this set of related techniques.

The literature is certainly full of attempts to combine these three techniques. Firstly, we note a number of applications of neural networks to a wide range of robot learning tasks including behaviour acquisition (Lin, 1991; Tani and Fukumura, 1994; Ziemke, 1996), navigation (Li, 1999; Li and Svensson, 1999; Morse et al., 1998), foraging (Nolfi et al., 1994), locomotion, classical conditioning (Mignault and Marley, 1997), exploration (Leow, 1998) and behaviour coordination. The application of neural networks to reinforcement learning problems is also popular (Ackley and Littman, 1990; Crites and Barto, 1996; Lin, 1992, 1993; Tesauro, 1992; Ziemke, 1996), as is the combination of BB and RL (most notably in Mahadevan and Connell (1991), but in many others including Maes and Brooks (1990); Mataric (1997); Shackleton and Gini (1997)). Specific combinations worth noting include the work of Lin (1991) who uses a backpropagation network to represent an RL system within a BB framework, and Dorigo and Colombetti (1994) who effectively combine BB, RL, and a Genetic Algorithm (Holland, 1975).

In their simplest form, neural units represent a fundamental sense-think-act coupling that is consistent with the BB philosophy. Neural architectures are naturally parallel, distributed and reactive, discouraging the excessive representation and manipulation of symbols. These also happen to be cornerstones of BB. Interestingly, Marvin Minsky, who was a key player in the neural field in the 1960s, later went on to pioneer some of the founding principles of behaviour based design too (Minsky, 1986).

CHAPTER 4

Real-valued RL: A Review

4.1 Introduction

Having introduced two important neural algorithms in the previous chapter, a range of existing techniques for representing and generalising over the action space of reinforcement learning problems can now be reviewed. The following section presents these techniques in an intuitive order that both groups approaches based on similarity and reflects our opinion of their relative sophistication. It should be noted that this does not always correspond to chronological order of publication. Of course where dependences exist between the accounts, this is made explicit.

For the following discussion it will be convenient to paraphrase the standard reinforcement learning problem in terms of figure 4.1. A number of approaches have already been outlined for generalising the state space, so now the emphasis is turned towards the action space. A key difference between the state and action space is that the input data that constitutes the state space is provided by the environment, whereas the action

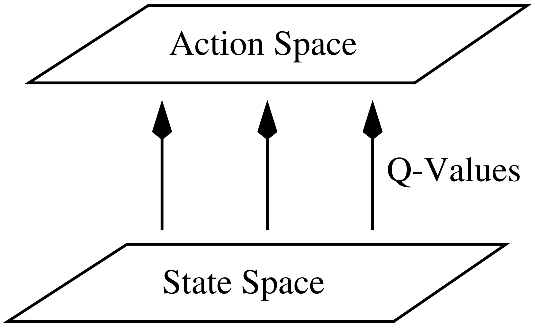


Figure 4.1: The basic reinforcement learning format.

space must be explored by the learning agent[1]. In games such as nought-and-crosses and backgammon the actions are discrete and fixed by the rules of the game (and the current state), so the action space is already in a suitable form for the application of the standard RL theory. Many researchers extend this idea to problems with large or continuous action spaces by simply handcoding this space as a small number of discrete actions. An example is the robot box pusher of Mahadevan and Connell (1991) in which the authors handcode five discrete actions: Move forward by a fixed amount, turn left on the spot by 22°, turn left on the spot by 45°, and two corresponding right turns. In principle, the robot could have made an effectively continuous range of movements, although it turned out that these five were sufficient for solving the problem they chose to address.

This kind of approach presupposes two criteria: Firstly that a fixed and small set of discrete actions will indeed be sufficient for maximising the reward signal, and secondly that such a set can be found beforehand. Some examples of problems where these criteria are unlikely to be met have already been suggested in chapter 1.

[1]This assumes that the input data is strictly independent of the agent's free parameters, which in general will not be true. This point is discussed later, but in the meantime we note that learning an appropriate representation of the input space is expected to be a more passive process than learning a representation of the action space.

4.2 Statistical Clustering

Because Mahadevan and Connell (1991) handcoded five discrete robot actions, they were able to reduce the scope of the generalisation problem to the state space alone. Their approach, which is now introduced for the purpose of benchmark comparison, is referred to in the original paper as *statistical clustering* although it clearly also resembles coarse-coding.

Each action, $a \in A_1 \ldots A_5$, is given a set of *clusters*, $C_{a,c_{1 \ldots k_a}}$ (k_a is the number of clusters associated with action, a), with each cluster representing a distinct region of state space[2]. Each cluster maintains its own Q-value, $Q(C_{a,c})$, which is intended to represent the expected reward of taking action a in areas of state space encapsulated by cluster $C_{a,c}$. Each cluster is represented as a sensory vector or *prototype*. A particular state-action pair (s,a) is added to a cluster $C_{a,c}$ (by moving the prototype towards s) only if s is already close to the prototype for that cluster *and* (s,a) yields a reward similar to $Q(C_{a,c})$. For a given state, s, actions are selected based on:

$$\sum_{c \in C_a} Q(c) \times probability(s \text{ falls within } c) \tag{4.1}$$

where $C_a = \{C_{a,1}, C_{a,2}, \ldots, C_{a,k_a}\}$, with the action which maximises this equation being selected with the highest probability. A key feature of this approach is that the state space is divided into regions of consistent reward *dynamically*, and *differently* for each of the five actions. However, the action space itself is represented explicitly with no provision or need for generalisation. This is in contrast to the kind of problems that are of direct interest to this thesis.

4.3 Coarse-Coding

Memory-based coarse coding was introduced in section 2.10.1 as a method of generalising directly over the Q-function. As in the work of Mahadevan and Connell (1991),

[2]The terms *state space* and *input space* will be used interchangeably throughout this document. *Sensory space* will also be used synonymously in this context. Conversely, the *action space* will sometimes be referred to as the *output space* or *motor space*.

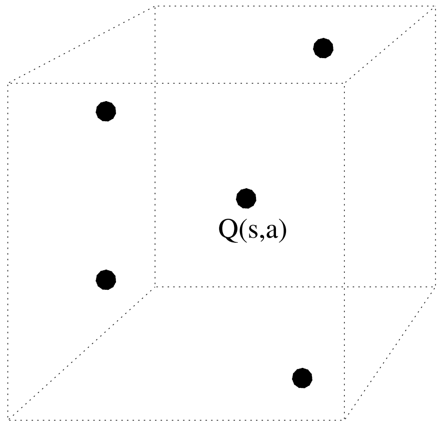


Figure 4.2: The combined state-action space of the Q-function is generalised by a number of prototypical Q-values.

the method uses prototypical Q-values (see figure 4.2) to represent the combined state-action space. Each prototype is a previously encountered state-action pair called an *instance* or, more generally, a *case*[3], and any position in the space can be approximated by an appropriate combination of prototypes. An example would be to consider a Gaussian distribution around each prototype, and build an approximation for the Q-value at a particular point (s,a) by a sum that is weighted according to the value of each Gaussian at (s,a). A new prototype is added if no existing prototype lies within a fixed distance of (s,a). Updating the representation given a position (s,a) and a reward value R involves updating the Q-value of each prototype towards R according to its distance from (s,a).

Santamaria et al. (1997) apply this approach to a version of the pendulum swing-up problem in which a two-dimensional input space must be mapped to a single-dimensional output space. Here, unlike Mahadevan and Connell (1991), both spaces are continuous. Although their account suggests that the approach is promising at least for this low dimensional problem, a number of drawbacks are apparent.

Because neither the input nor action space is represented explicitly, selecting the best action for a given state becomes a search problem. In other words there is no direct

[3]This class of techniques is also known as *instance-based* or *case-based* approximation.

way to interrogate the representation for the optimal action given a state as input. An overhead is essentially introduced because the representation of the space is not ideally suited to the way in which that representation is to be accessed. *Case-based* coarse coding attempts to address this by separating out the representation of the state and action spaces. This time the state space is represented by a set of prototypical states (not state-action pairs) which the agent has experienced. These are called cases. For each case, a number of fixed actions are available, and Q-values are maintained for each of these actions within each case. Retrieving the Q-value of any state-action pair, (s,a) then requires a three step process:

Firstly, each case estimates its own Q-value by using a weighted sum of the Q-values of each action within that case, with the weights depending on the distance of the action (in action space) from the query action a. Secondly, the relevance of each case is determined according to its distance, this time in state space, from the query state s. Now the set of Q-values in phase one are summed according to the weights of each case determined in phase two. This gives the Q-value of the query state-action pair, (s,a). The idea is similar to the simpler instance-based technique, except that the action space is separated from the state space, and the two are treated one at a time. One advantage of this approach is that selecting an action can be simplified to first identifying the nearest case to the current state, and then simply selecting the action within that case which has the highest Q-value. However, these actions are static and discrete. If we wish to interpolate over the action space as well as the state space in selecting an optimal action, then we are again faced with a search problem in the action space.

Another drawback of coarse-coding is that although prototypes may be created dynamically, they are static within the state-action space. In a non-stationary environment, this could result in an inefficient representation with aggravating implications for the problem of searching the space for optimal actions. Hence one needs to consider how instances or cases may be removed, as well as added and updated.

4.3.1 Nearest Neighbours

For consistency it is worth referring to the work of Moore (1990), in which a number of *nearest neighbour* function approximators are discussed, all of which fit into the same memory-based class as the instance-based and case-based algorithms outlined above. The idea is again to represent a function (in our case, we are interested in the value function of an RL problem, but clearly the idea is more broadly applicable) with a set of prototypical values. Moore outlines a number of ways to interrogate such a function which include interpolation between prototypical values (see 'Shepard's Interpolation' in Moore (1990)), simply taking the value of the nearest neighbour (or the mean of a set of N nearest neighbours), and using Gaussian Kernels over the prototypes to achieve a continuous function approximation. However, as with instance-based and case-based methods, this is only a representational framework, and clearly identifying optimal actions for any given state requires a significant search overhead.

In chapter eight of Moore (1990), the process of searching for optimal actions is achieved by randomly generating a number of candidate actions and then evaluating how good each of those actions are likely to be given the target behaviour and a set of previously encountered state-action-behaviour prototypes. The evaluation process generally looks for actions which are likely to yield the desired behaviour for the current state. However, when the action space is being explored, actions are selected at distances from prototypes that are proportional to the distance of the prototype from the desired behaviour.

Moore's work is not actually concerned with RL, and is included here for completeness since he addresses generalisation in real-valued action spaces.

4.4 Kohonen mapping the state-action-reward space

In a similar style to Santamaria et al. (1997), Touzet (1997) uses a Kohonen network to map the combined state-action-reward space in a task that requires a robot to learn to move forward through an environment, avoiding obstacles en route. Each unit in the map can be thought of as a prototypical instance of a state-action-reward tuple. Touzet uses a reward signal which takes one of the three values: $\{-1,0,1\}$. For a given state, s, the optimal action, a, is selected by first finding the unit with the smallest distance on the partial weight vector: $[s \ldots r]$, with $r = 1$ (the optimal reward). The weights that comprise the missing action dimensions are then used to define a. Care must be taken when selecting an appropriate winning unit to balance the unit's distance to s with the distance to $r = 1$, otherwise successfully matching a large state vector may result in a poor match to $r = 1$, and hence the selection of a suboptimal action. If the reward is continuous with no fixed upper limit then a search for an optimal action is required because each unit must now be considered with respect to both the likelihood of that unit representing the current state, *and* the estimated return offered by the action weights of that unit. This yields an approach similar to that described in section 4.2.

Although not made explicit, it is presumed that Touzet achieves exploration by occasionally perturbing the proposed action, $a \rightarrow a'$, and then updating the Kohonen map to the tuple $< s, a', r >$. This would then allow the action space to be explored which is clearly necessary to uncover the rewarded regions. According to the comparisons made in the paper, using a Kohonen network to map the combined state-action-reward state is both effective and efficient, with a learning speed that improves on a number of other approaches including:

- Explicit representation of state and action space (with the spaces made discrete beforehand in a manner similar to Mahadevan and Connell (1991)).
- Statistical Clustering (also from Mahadevan and Connell (1991)).
- Generalisation based on backpropagation.

However, the problem considered is relatively simple, and scaling issues are not addressed. For example, although the input space is eight dimensional (corresponding to the number of sensors on the robot), there is much dependency between the sensor values. There is also likely to be irrelevancy on two backward facing sensors because the robot only moves forward. The reward signal is also simplified by considering essentially only binary values corresponding to whether the situation has 'improved' as a result of the last action. The simplicity of the problem is reflected by the fact that a

Kohonen network of only 4×4 units is used and the neighbourhood function consists of updating just the four immediate neighbours.

Given that the state and action dimensions of the combined state-action-reward space will be independent, then assuming just a single state variable and a single action variable, the intrinsic dimensionality of this space must be at least two. In general, combining state, action and reward spaces may be asking too much of the generalisation capabilities of a two-dimensional Kohonen map. However, Touzet is successful in showing that the SOM is a potentially useful candidate for generalisation over continuous action spaces.

4.5 The Motoric Map

The Kohonen map is also used by Wedel and Polani (1996) to generalise over both the input and action spaces. Their approach is based on the *Motoric Map* of Ritter et al. (1990) which is briefly described here. A Kohonen network is first used to map just the input space so that each unit represents a region of that space. Associated with each unit of the map is an action weight vector and a Q-value[4] which are intended to represent the optimal action for that unit and the estimated expected return of taking that action respectively. The optimal actions for each unit could be provided by an explicit teacher or learned by a stochastic search of the action space (see figure 4.3). As in the model proposed in chapter 5, and following the notational convention outlined in chapter 2, the Q-values are really estimates of the expected return under a particular policy, π, which depends on the search of the action space. In this sense, the value estimation technique is strictly more similar to SARSA then Q-learning.

Ritter et al. (1990) successfully apply a Motoric map with a single action variable to the cart-pole-balancing problem[5]. The action space is randomly searched using the

[4]Note that because there is only a single action for each state, the estimates of expected reward represent a *partial* Q-function or, more strictly, the value function, V. However, the partial Q-function notation is preferred for consistency with the rest of the models considered in this thesis. In other words we choose to interpret this model as the special case of the more general model in which there are any number of actions associated with each state.

[5]The problem consists of a cart which can move backwards and forwards along a horizontal track. A pole is mounted on the cart by a hinge at the base, so that it can rotate in the dimension of freedom

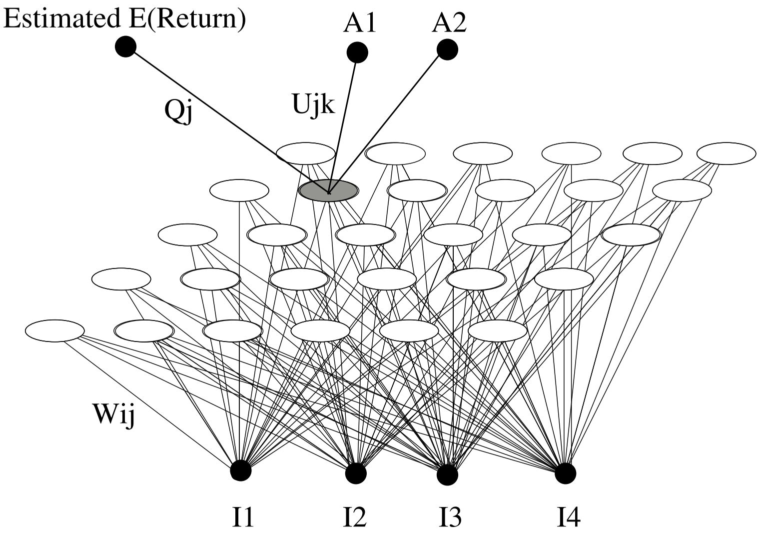


Figure 4.3: The Motoric map. The Kohonen network maps the input space. For a given input vector $\mathbf{s} = [I_1, I_2, I_3, I_4]$, a winning Kohonen unit, j, is selected with minimum weight distance: $|\mathbf{s} - \mathbf{w^j}|$. The winning unit maintains a separate action weight vector $\mathbf{a} = [U_{j1}, U_{j2}]$, and an estimate of the expected return of the pair $(\mathbf{s}, \mathbf{a})$, Q_j. Action $\mathbf{a}$ can be taken and the resulting reward used to update Q_j *or* a perturbed version of $\mathbf{a}$ can be taken (say $\mathbf{a}'$, generated for example by a Normal distribution around $\mathbf{a}$) and, if it yields higher reward than Q_j, unit j updated towards this perturbed action on the relevant output weights. The policy being evaluated is implicitly defined by the actions attached to each state.

reward signal to find an appropriate action for each unit of the map (as described in figure 4.3).

Although the Motoric map is successfully applied to the cart-pole-balancing problem, Wedel and Polani (1996) suggest that a random search of the action space is likely to be prohibitively slow in higher dimensions. With the intention of addressing scalability, they adapt the Motoric map in the following way: Instead of taking just one perturbed action, $\mathbf{a}'$, and moving the winning unit of the map towards that point in the action space if it happens to yield an improvement in reward, they take a number of sample actions around $\mathbf{a}$, and use the elicited rewards to parameterise a Gaussian approximation of the reward function around $\mathbf{a}$. Given this local model of the reward function, the *direction* of highest reward can be estimated, which then requires a further line search

of the cart. The aim is to keep the pole upright by moving the cart to the left or right according to the angle of the pole. The input space usually consists of the angle and angular velocity of the pole along with the position and speed of the cart, while a single action variable represents the horizontal force to apply to the cart. The reward signal is based on the angle of the pole. See Peng and Williams (1996) for an example.

to yield an optimal action. Because the Gaussian model is only an approximation, the algorithm is iterative with new Gaussian approximations being made as the unit's action weights are updated, and new perturbed actions sampled. Because fitting the Gaussian requires finding an appropriate covariance matrix, Wedel and Polani (1996) call this algorithm *covariance learning*.

The basic idea of the Motoric map is taken further as part of this thesis, with the idea of using a second Kohonen map to represent the output space explored in some detail. The suggestion of Wedel and Polani (1996) that random exploration of higher dimensional action spaces may be problematic is also investigated within the context of tasks involving multidimensional state and action spaces. The idea of using a SOM is also explicitly extended to situations where the reward is delayed (i.e. considering discounted returns rather than immediate rewards), and applied to systems involving multiple behaviours.

Covariance learning appears to have a number of problems. The algorithm is naturally off-line, with a preference for sampling the points around $\mathbf{a}$ independently of control. For example, when the only way to sample the reward function is to actually take an action, the environment state will tend to change. Collecting enough samples at each state to generate a reasonable Gaussian may require many independent visits to a state, during which time the reward function may have changed (if the environment is non-stationary). Also, in their experiments, the Kohonen network first maps the input space after which the input weights of the map are frozen. Given this fixed state representation, the action space and reward function is then explored. Because changing the actions taken in each state will generally result in a different input distribution (thus rendering the input mapping obsolete), the two phases are iterated, but it turns out that performing both the input mapping and action space exploration simultaneously is not successful in their account. Interestingly, this contradicts results obtained from experiments performed as part of this thesis (see chapter 5). Thirdly, their attempts at speeding up learning by updating each unit's covariance matrix from those of its nearest neighbours is unsuccessful. Again, this is in contrast to experiments performed as part of this thesis in which units will be seen to be able to benefit significantly from reward information gained by their neighbours (chapter 5).

Although Wedel and Polani (1996) claim that their approach should improve the performance of the basic Motoric Map in high dimensional action spaces, they offer no results to support this claim, and perform no experiments involving action spaces of dimension greater than two[6]. However, the idea of using a simple model to locally approximate the reward function, and then using this model to derive a potentially lucrative direction for exploration appears to be sound. The key feature of both the Motoric map, and its covariance learning extension is that a continuous action space is explored and generalised adaptively. This is in marked contrast to the approach of Mahadevan and Connell (1991) in which a small number of discrete actions are fixed beforehand.

4.6 MLP generalisation

The Multi-Layer Perceptron (MLP) with its backpropagation training algorithm provides a powerful alternative for the representation and generalisation of both the state and action spaces. In Tesauro's *TD-Gammon*, a feedforward network with a single hidden layer of units was trained to map states (or board positions) to the *value* (probability of winning) of that state. In this way the network was used to generalise over the state space, with the actions fixed by the rules of the game. However, in cases where the environment model is unknown, this approach is rendered unsuitable because even if the value of each state is known precisely, there is no way of knowing which action will result in a transition to the best states.

One approach to MLP generalisation advocated by Lin (1993) is shown in figure 4.4. The principle is similar to that adopted by Tesauro, except that now a separate value function is maintained for each action. Selecting the best action in a given state is simply a matter of identifying the network with the highest output. This allows the state space to be generalised, but the obvious drawback is that the action space must be discrete (and preferably small) suggesting a handcoding approach similar to that of Mahadevan and Connell (1991).

[6]The actual application involves moving across a two-dimensional plane to a goal. The input space encodes the current coordinate of the agent, the output space comprises a continuous range of movement vectors, and reward is given proportionally to the distance of the agent from the goal.

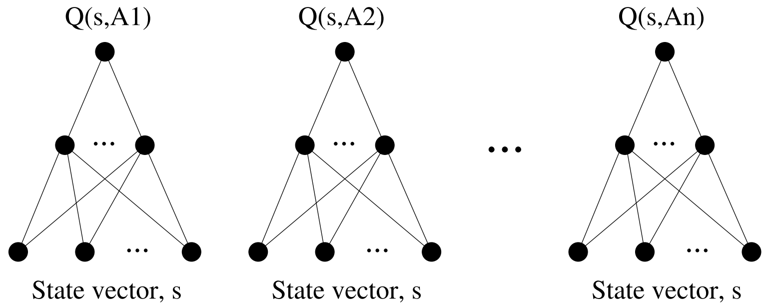


Figure 4.4: Lin's *QCON* model where a separate backpropagation network for each action $(A_1 \ldots A_n)$ learns to map states to Q-values under those actions.

To address adaptive generalisation over the action space, Touzet (1997) considers a slightly different approach in which only a single backpropagation network is used. As before, the inputs to the network code the current state, but this time the outputs directly code the actions rather than Q-values. The approach adopted is as follows: Each action is broken down into individual variables. For example, an action corresponding to a left turn on a differential drive two-wheeled robot is broken down into a left wheel action and a right wheel action. For each of these elementary variables, a second *antagonistic* variable is introduced as suggested by figure 4.5. Only one action from each agonist/antagonist pair is selected at a time. The output of each action unit is interpreted as the speed of that motor in that direction. At each state, the elementary action of each agonist/antagonist pair with the highest value is selected. This relies on their second interpretation of the output of each action unit as the suitability of that action. Each action variable is perturbed by a small amount and then the perturbed action taken. A reward of 0,1 or -1 is received with the following consequences:

❶ **Reward = 0**. Do nothing.

❷ **Reward = 1**. Train the network (only on the weights to the selected unit of each agonist/antagonist pair) towards the perturbed action.

❸ **Reward = -1**. Train the network (on all weights) to the current outputs with the agonist and antagonist outputs interchanged.

Although the algorithm performs satisfactorily on a simple problem, the approach is

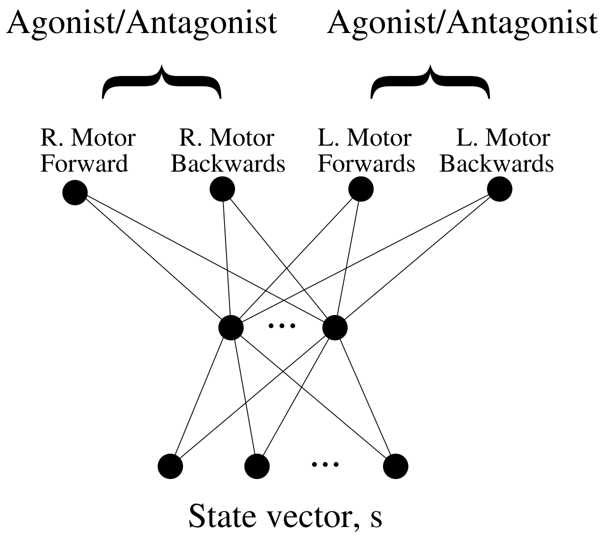


Figure 4.5: Touzet uses the MLP to generalise over both the state and action spaces. The approach relies on providing each elementary action with its complement, with the two then competing for dominance in each state.

somewhat untidy and relies on some odd assumptions relating the suitability of an elementary action such as 'left motor forward' to the strength of that action (i.e. how fast to move the left motor forward). A lack of generality is also imposed by the tertiary reward signal, and the need for complementary actions to be coded at the output units. There is also significant divergence from the theoretical groundings of Q-learning (or indeed any of the value estimation techniques discussed back in chapter 2), with a somewhat arbitrary looking learning rule in ❸. However, the approach is noted for its application of the MLP to generalisation over both continuous state and action spaces.

Touzet subsequently compares this technique to that described in section 4.4 where he uses the Kohonen network to generalise over the combined state-action-reward space. However, the comparison fails to provide the detailed results or analysis necessary for identifying the underlying strengths and weaknesses of these two very different approaches to generalisation. In particular, his argument depends more on the contingent properties (as opposed to the necessary ones) of each approach because only one small problem is considered. For example, the Kohonen map is concluded to be more efficient, but the non-linear generalisation potential of backpropagation (which should be considered compensation for the heavier duty algorithm) is never explored. Brief allusion is made to the difference between the local Kohonen update rule and the non-local backpropagation learning rule, but the discussion is not pursued to a satisfactory conclusion. These issues are taken up in chapter 8.

4.7 The CRBP algorithm

Prior to Touzet's work, Ackley and Littman (1990) proposed a similarly motivated algorithm called the *Complementary Reinforcement BackPropagation Algorithm* or *CRBP*. The algorithm is again built around the backpropagation network and the basic architecture is shown in figure 4.6. At each time-step, a state vector, s, is presented to the network. The activation of each output node, O_k, is interpreted as the probability of a corresponding binary variable, B_k, being set to 1. The binary vector, B, then represents the actual action to be taken. The reward signal is also binary, with a value of 1 corresponding to a 'good' action, and a value of 0 to one which should not be reinforced. If the reward for the current state-action pair (s,B) is 1, then the network is updated towards that pair, so that the network output vector, O, will be more likely to yield the action vector B next time s is presented. If the feedback is negative, then following the approaches of (Barto and Anandan, 1985; Anderson, 1986; Williams, 1992; Ackley, 1989), the network is updated towards the pair $(s, 1-B)$, thus discouraging B on future presentations of s.

The approach works well on a number of simple tasks including the *n-majority* problem in which a single binary output variable must take the value 1 if more than half the binary input values are 1, and 0 otherwise. Within the context of this and other similar problems, the generalisation capabilities of a backpropagation network are compared favourably with a table look-up approach in which an action is stored explicitly for each possible input vector. They succeed in showing that while for small state spaces an explicit lookup table learns quicker, as the number of distinct states increases, the scalability of the backpropagation network is superior. More interestingly, they also demonstrate how the network is capable of generalising over the action space. They consider a set of learning tasks in which there are a number of correct outputs for each input. For example, the *bit-count* problem which requires that the output has the same number of 1s and 0s as the input, with the position of particular values being irrele-

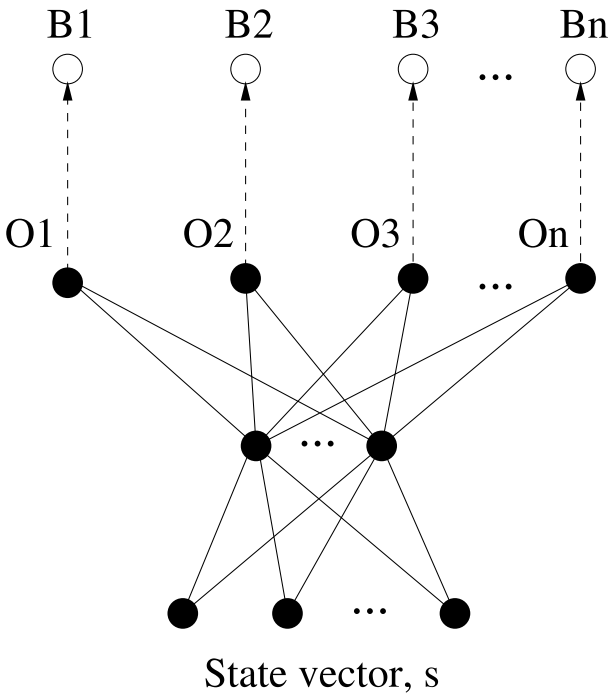


Figure 4.6: The Complementary Reinforcement BackPropagation (CRBP) architecture of Ackley and Littman (1990). Each output, O_k, of a backpropagation network is interpreted as the probability that a corresponding binary variable, B_k, is set to 1. If the reward is positive then the network is trained towards the pair (s, B), otherwise it is trained towards the complement $(s, 1 - B)$.

vant. They found that the network succeeds in finding a more compact set of actions than the table look-up approach, with each action being used for a range of different inputs. They also report robustness to 'overfitting' (involving networks with more free parameters than are necessary), but note that for some problems the network can be outperformed by a simple table lookup approach, particularly when the categorisation is not linearly separable in the input space (and therefore requires the learning of two layers of weights).

The scalability of learning to the complements of punished actions is unclear, and indeed the use of a binary reward signal is itself restricting since it requires that the actions can be qualified as either 'good' or 'bad'. As with Touzet's work, the authors have again chosen to make a significant departure from the underlying reinforcement learning theory, with the effective abandonment of the central concept of estimating expected reward. However, it is straight forward enough to adapt the algorithm so that a Q-function is maintained and used to turn a continuous scalar reward signal into the binary reward signal necessary for driving the learning here. In fact this is the basis of both Gullapalli's SRV units (see section 4.9) and the experiments of chapter 8, with the latter considering the use of MLP based techniques for generalising over the action space in much more detail.

One significant drawback of CRBP is the requisite that actions be represented as binary vectors, thus precluding generalisation in continuous action spaces. This is addressed in both Ziemke's application of CRBP to robot learning and Gullapalli's SRV units, both of which are now introduced.

4.8 Continuous space CRBP

An interesting application of CRBP to robot learning is found in Ziemke (1996). A simulated Khepera robot (Michel, 1996) is trained to wander around a two-dimensional environment avoiding obstacles en route. The circular robot is driven by two wheels — one on either side — each of which may take any value in the continuous range $[-1,1]$ with 1 corresponding to a maximum forward speed, and -1 to a maximum reverse speed. By setting the wheel motors to appropriate speeds a wide range of different movements can be achieved comprising both forward and reverse turns with $0 \leq$ radius $\leq \infty$ and $0 \leq$ speed $\leq$ max. speed. The robot also has six distance sensors equally spaced around the front of the agent, each with its own receptive field. The activity of these continuous valued sensors increases with the proximity of an obstacle within the receptive field. For clarity, the robot is shown in figure 4.7 and its range of movement in figure 4.8. While the simulator cannot hope to perfectly reproduce the dynamics of the real Khepera robot (Mondada et al., 1993), the simulator still provides a useful tool for robotic experimentation.

In an approach similar to that of Mahadevan and Connell (1991), Ziemke begins by handcoding three discrete actions: Move forwards, left turn, and right turn. These three actions are coded using three binary output units corresponding to B_1, B_2 and B_3, in figure 4.6. The binary reward required by the CRBP algorithm is generated as follows: If there is any sensor activity, then providing that activity is less than that

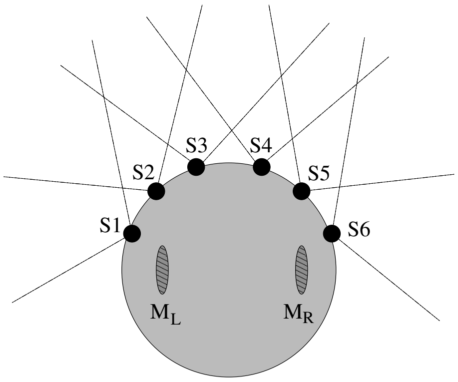


Figure 4.7: The Khepera robot used in simulation by Ziemke (1996). The two motors are labelled M_L and M_R, and the six distance sensors (along with their receptive fields) as $S1 \ldots S6$.

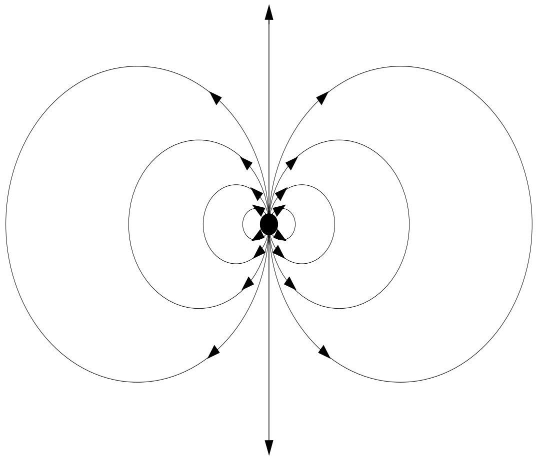


Figure 4.8: By appropriately setting the speeds of the two wheels, the robot can be made to turn forwards or backwards with $0 \leq$ turning radius $\leq \infty$ and a continuous range of speeds.

recorded at the previous time-step, reward = 1, else reward = 0. If there is no significant sensor activity, then reward = 1 if and only if the robot is moving forwards. By learning *towards* actions which yield positive reward and *away* from actions yielding zero reward, the agent learns to wander around the environment without colliding into obstacles.

The second experiment is more interesting since it specifically addresses the representation of continuous action spaces. This time, instead of the binary output units, $B_1 \ldots B_3$, being used to represent three distinct actions, the actual output units of the network, O_1, O_2, are used to represent the motor speeds directly. Since these units are continuous, the CRBP network is now effectively performing generalisation over both the continuous six-dimensional input space *and* the continuous two-dimensional action space. Learning takes place in the usual way, with a positive reward resulting in an update of the network towards the pair (s, B), and a negative reward in an update towards $(s, 1 - B)$, where B is the vector of binary values derived from O according to figure 4.6.

Again, the agent learns its task of wandering and avoiding obstacles, but as with the work of Ackley and Littman (1990) the scalability of the approach is unclear. Of particular concern is both the generality and scalability of using binary rewards to update continuous variables towards binary values. In Ziemke's experiment, it was possible to solve the problem with actions involving extreme motor values. An interesting and important experiment would test the ability of the architecture to learn a task requiring intermediate real-valued actions. It is not clear how the proposed architecture would achieve this given the binary reward signal.

Ziemke (1996) is actually primarily concerned with investigating a novel backpropagation architecture[7], and is not investigating CRBP directly but rather using it as a tool for moulding RL into a form suitable for use with backpropagation. He acknowledges that CRBP may not be suitable for more complex applications, and does not pursue an analysis of the algorithm itself. However, in the context of this thesis it represents an applied approach to generalisation in which a backpropagation network is used to map states directly to actions within continuous spaces, without explicit estimation of a value function. While this circumnavigates the problem generated by the lack of an environment model, the generality of the approach is questionable. Again the drawbacks of CRBP are confirmed as: Significant divergence from the RL theory with no explicit representation of a value function (or Q-function), a possible lack of generality when considering genuinely real-valued functions, and the potentially debilitating constraints of immediate and binary reward.

4.9 SRV units

Recall that backpropagation was used in Tesauro's *TD-Gammon* to generalise over the state space with an environment model used to select optimal actions from a predefined and discrete set. Lin (1993) addressed the problem which arises from not having an environment model by using a separate MLP to represent the value function of each action, but this required that a small set of discrete actions were defined beforehand in a manner similar to the box pushing robot of Mahadevan and Connell (1991). To avoid

[7] In which the first layer of weights are determined by the activations of the hidden units.

this drawback, Touzet (1997) used a single network to map states directly onto agonist/antagonist pairs of actions, with a binary reward signal responsible for reinforcing the network to either the agonist or antagonist. The same idea is apparent in the CRBP algorithm of Ackley and Littman (1990) with the binary units, $B_1 \ldots B_n$, effectively fulfilling the role of agonist and antagonist. But the simplification of the reward signal as both immediate and binary represents a significant constraint on the underlying theory presented in chapter 2, while the indulgence of discrete and even binary actions is potentially very restricting indeed. Ziemke's minor modification to CRBP for learning and generalising over continuous action spaces appears to lack both scalability and generality. As well as being divergent from standard RL theory, this last approach also lacks both empirical and analytical justification, as well as the confidence of the author himself.

The first satisfactory approach to generalisation over continuous action spaces is found in Gullapalli (1990). His approach is based on backpropagation but, unlike the MLP models introduced so far, exhibits the kind of generality and faithfulness to the RL theory that inspires confidence. Gullapalli's technique is considered in detail in section 8.3 where it is compared with a new architecture that is introduced in the next chapter. For this reason, only a brief overview is given here, with a full analysis postponed until chapter 8.

The problem that Gullapalli sets is that of generating an optimal mapping from real-valued input vectors to a real-valued output guided only by a scalar reward signal. His approach is based upon his Stochastic Real-Valued (*SRV*) unit of figure 4.9. The idea is simple. The system is fed two inputs (the model can clearly be generalised to any number of inputs) and an output O is generated using a Normal distribution. The mean for this distribution is generated by a single linear hidden unit, $H1$, whose output is the weighted sum of its inputs (including a bias). The second hidden unit, $H2$, which is also a linear unit, generates the estimated expected reward, $E(R)$, of the SRV unit for the given input stimulus.

Gullapalli restricts the reward signal to the range $[0,1]$, so $E(R)$ also lies in this range. The variance of the Normal distribution is generated according to $E(R)$ and the following principle; if $E(R) = 1$ then the unit is behaving optimally, and the variance is zero.

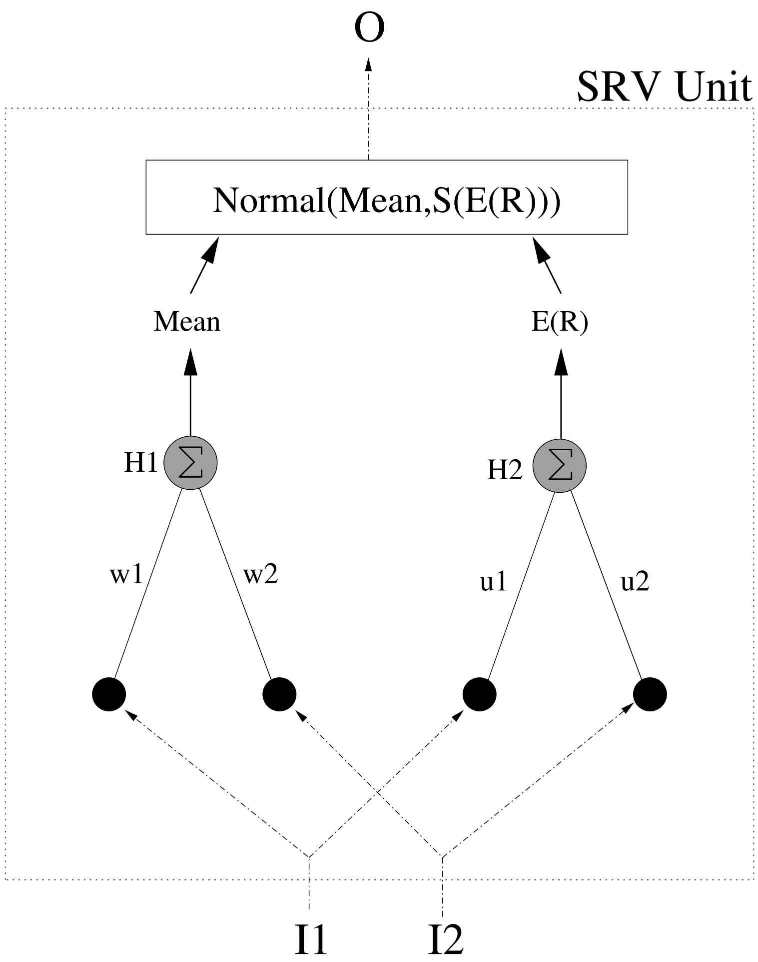


Figure 4.9: Gullapalli's SRV unit. See text for a description.

If $E(R)=0$ then the SRV unit is behaving very poorly and variance is set high. The variance is actually calculated from $E(R)$ using a monotonically decreasing function, S. After an input vector has been presented to the SRV unit, and the action generated by the Normal distribution has been taken, a reward, R, is elicited from the environment. Now hidden unit $H2$ learns to produce R for the current input vector by backpropagating the error $R-E(R)$ through the linear unit according to the Widrow-Hoff learning rule (Widrow and Hoff, 1960):

$$u_i(t+1) = u_i(t) + \beta(R - E(R))I_i(t) \tag{4.2}$$

This is just a special case of the more general backpropagation update rule where each weight is updated negatively proportionally to the gradient of the Error function with respect to that weight. Here the error function is implicitly defined as $\frac{1}{2}(R - E(R))^2$.

Unit $H1$ is updated in much the same way except that its target is $O \times (R - E(R))$. Hence if the actual reward is an improvement on the estimated expected reward, then the mean of the distribution is pushed towards O proportionally to how much better it is, otherwise the mean is pushed away from O. The actual update rule used for unit $H1$ is:

$$w_i(t+1) = w_i(t) + \alpha(R - E(R))\left(\frac{O(t) - mean(t)}{\sigma(t)}\right)I_i(t) \tag{4.3}$$

The intuitive idea behind the SRV is simple. For each input vector a mean and variance is computed, with the variance linked to how far the estimated expected reward is from a theoretical maximum. In this way the variance provides the random *exploration* needed to discover useful actions. An output is then generated according to a normal distribution based on this mean and variance. If the output is good, then the mean is pushed towards the output otherwise it is pushed away from it. In each case, the estimate of the expected reward is updated. Learning proceeds in this way until the SRV unit produces the correct output for each input, at which point the variance of the distribution will hopefully be zero.

Gullapalli tests an SRV unit on the AND problem:

I1	I2	Output
0.1	0.1	0.1
0.1	0.9	0.1
0.9	0.1	0.1
0.9	0.9	0.9

with the reward at each time-step defined as $1 - |Error|$ where the $Error$ is the difference between the target output and the actual output of the SRV unit. The single unit easily learns the mapping in around 2000 time-steps, where a time-step corresponds to a single processing cycle which involves: the presentation of a single input vector, taking the action proposed by the SRV unit, an environmental reward, and the application of the weight update rule.

Clearly this architecture will be unable to solve linearly inseparable problems since the units H1 and H2 only have a single layer of adaptable weights. Gullapalli extends the model to networks of SRV units in order to tackle the XOR problem, although results are somewhat disappointing. However experiments performed as part of chapter 8 and based on the SRV concept of maintaining an action plus an estimate of expected reward for that action, appear to show that the principle is sound and probably more reliable than Gullapalli's initial results would suggest[8].

Although the results of the SRV approach are not particularly impressive, a number of desirable properties will lead us to explore and improve this algorithm later. In particular, Gullapalli's model fully embraces both continuous state and action spaces and a continuous reward signal, and can in principle be extended to multiple outputs and delayed rewards. The RL theory is also adhered to in that an explicit estimate of expected (and potentially discounted) reward is maintained. Issues such as scalability, generality, applicability, and a review of generalisation properties are offered in chapter 8 with reference to a number of alternative models — in particular the Kohonen map.

[8]The way in which Gullapalli integrates a network of SRV units with the backpropagation learning rule may be faulty. See chapter 8.

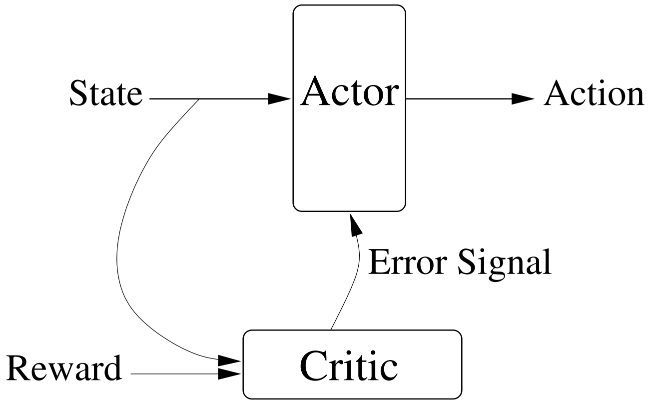


Figure 4.10: The Actor-Critic model applied to early reinforcement learning problems. The actor maps input vectors to actions, while the critic maintains the value function which is used to update the actor.

4.10 Q-AHC

Another approach to dynamic real-valued actions that is worth discussing is *Q-AHC* (Rummery, 1995). Rummery's model is based on the Adaptive Heuristic Critic (AHC) or *actor-critic* model of (Barto et al., 1983; Sutton, 1984). The general scheme is shown in figure 4.10. The *actor* normally has a number of discrete actions available to it and maintains probabilities of taking each action in each state. The *critic* maintains a separate representation of the value function which is used to update the likelihood of taking each action in each situation based on environmental feedback. This idea has now been somewhat eclipsed by approaches such as Q-learning which simplify the issue by combining the representations that underpin the two processes of estimating expected reward and the selection of actions.

Inspired by Williams (1988), Rummery (1995) adapts this basic actor-critic model to deal with real-valued actions. In a manner similar to Gullapalli (1990), actions are generated using a Gaussian distribution, with a mean and standard deviation calculated as a continuous function of the state information. If the action yields a reward that is greater than the expected reward (as maintained by the critic), then the mean and variance for that state are updated according to the gradient of the log of the Gaussian distribution, so as to make that action more likely on future occasions. This involves moving the mean *towards* the action if the reward received is greater than the estimated reward, and away from the action otherwise (only single dimension action vectors are considered at this stage). Similarly, the standard deviation is either increased or decreased to make the explored action more or less likely (as appropriate).

Rummery uses backpropagation networks to represent the critic as well as the functions, $\mu(s)$ and $\sigma(s)$. The augmented scheme is shown in figure 4.11. This model is then extended further to incorporate multiple Q-AHC modules with each module having jurisdiction over a disjoint region of state space. For a given state, each AHC module generates a proposed action, along with the expected return of taking that action in that state. The module predicting the highest return is then selected. In this way the action space is explored in parallel and there is an implicit notion of competition between AHC modules for control of the agent. However, the results reported are somewhat disappointing according to Rummery, because other models using a fixed hand-coded action set outperform Q-AHC on the tasks attempted[9]. He also reports particular problems with multiple Q-AHC modules learning the same actions even though there may be other better actions still to be found — although it is noted that this redundancy may be artificially alleviated by restarting some AHC modules with new parameters if necessary. Rummery suggests a number of reasons for poor performance, including an insufficient number of hidden units in the MLPs, and suboptimal local maxima being found by the stochastic hill-climbing mechanism used to search the action space (recall the way in which the mean of the Normal distribution is updated).

However, as with the SRV unit of Gullapalli (1990), this approach, based upon the MLP and backpropagation, will turn out to be useful in chapter 8 when distributed approaches to generalisation are compared with local function approximators. In the meantime we note that Rummery's model is one of the most sophisticated and general approaches to dynamic real-valued action generation in multiple dimensions, with the key concept of estimating expected return (real-valued and fully discounted) at the heart of the model.

[9]Some of these tasks are extended to multiple dimensions, but the essence of the model is the same.

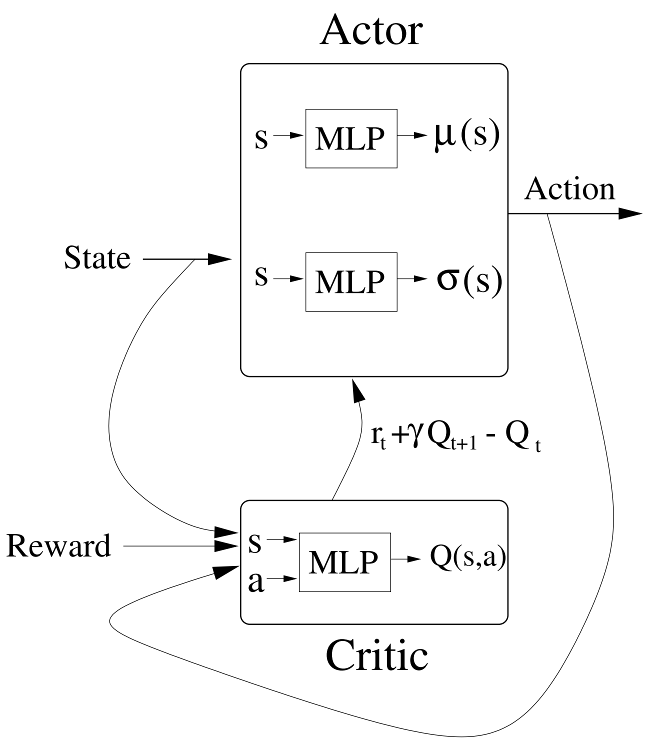


Figure 4.11: The Q-AHC model of Rummery (1995). The actor generates real-valued outputs based on a Normal distribution parameterised by a mean and standard deviation, each of which is a function (MLP) of the state information. The critic maintains the Q-function (another MLP), which is updated towards the return using any appropriate method (Rummery uses Q-learning with eligibility traces — i.e. Combining Q-learning and $TD(\lambda)$). The error between the predicted return and the actual corrected one-step return is used to decide whether the proposed action should be reinforced or not. The weights of the actor are then updated to make the proposed action more or less likely, as appropriate.

4.11 CMAC

The final method considered is that of Prescott (1994) who combines the Cerebellar Model Articulation Controller (CMAC) of Albus (1975) with a continuous action-space exploration strategy proposed by Williams (1988). The problem Prescott considers is that of using RL to train a simulated robot to move around an environment avoiding collisions with obstacles. The simulated robot has three range finders, each of which returns a real-valued distance along the line of that sensor to the nearest obstacle. The output is interpreted as a real-valued speed and turning angle. Reward is real-valued and delayed, and is made up of two components — one rewarding large translational speed (but not rotational speed), and one punishing collisions with obsta-

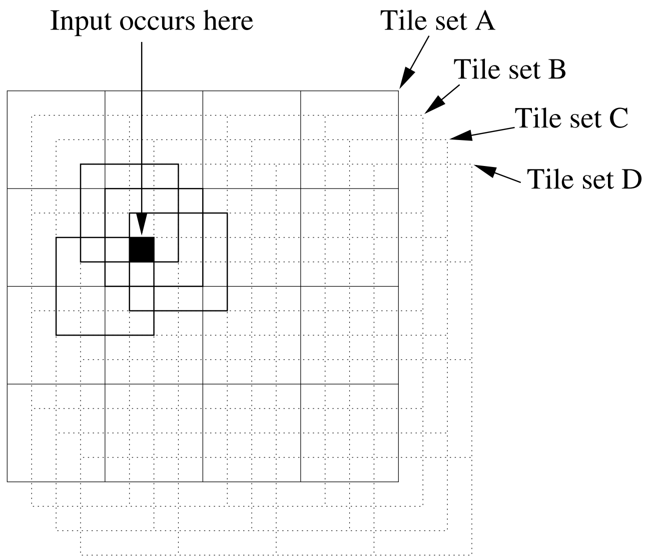


Figure 4.12: The CMAC approach to tiling the state space adopted by Prescott (1994). In this example the space is tiled by four offset, overlapping tile sets, A, B, C and D. Each tile set happens to consist of 4x4 individual tiles. Any point in the continuous two dimensional input space can be recoded by considering the four tiles (one from each tile set - outlined in bold) in which the point lies.

cles. Hence the agent must learn a mapping from a continuous three dimensional state space to a continuous two-dimensional action space in order to maximise a real-valued, delayed reward signal.

The input space is tiled using a CMAC (Albus, 1975) (without hashing) as shown in figure 4.12. In Prescott's model the state space is three dimensional and five tile sets are used, each containing $5 \times 5 \times 5$ individual tiles with each tile set offset by $\frac{1}{25}$ of the maximum size of each dimension of the space. Any three dimensional real-valued input, s, is converted to a new vector, $\boldsymbol{\phi}(s)$, with each element of the new vector corresponding to one of the $5x5x5x5 = 625$ tiles. For any input, all elements of the new vector are set to zero, except for the five elements which correspond to the five tiles (one from each tile set) in which the input occurs. Each of these elements are set to $\frac{1}{5}$ so that the total activity in the new vector sums to unity. This process of re-mapping a continuous three-dimensional vector to a binary 625 dimensional vector is advantageous because the hope is that the elements of the new vector can be combined linearly (and therefore learned quickly) to produce either actions or estimated expected returns.

Prescott's full model is similar to the actor-critic model introduced in section 4.10. For each state, s, the critic estimates the expected return of taking the action proposed by the actor. The output of the critic, $\hat{V}(s)$, is simply a linear combination of the elements of the mapped input vector, $\boldsymbol{\phi}(s)$:

$$\hat{V}(s) = \mathbf{w}^T \boldsymbol{\phi}(s) \tag{4.4}$$

For every state, the critic is always updated towards the return following the action:

$$r + \lambda \hat{V}(s') \tag{4.5}$$

where $\hat{V}(s')$ is the output of critic for the successor state. Because the output of the critic is a simple linear combination of the elements of the vector, $\boldsymbol{\phi}(s)$, this update is a straightforward matter involving first defining a squared error term,

$$E = \frac{1}{2}\big(\text{target} - \text{actual output}\big)^2 \tag{4.6}$$

then calculating the gradient of this error with respect to each weight, w_i,

$$\frac{\delta E}{\delta w_i} = \big(\text{target} - \text{actual output}\big)\boldsymbol{\phi}(s)_i \tag{4.7}$$

and then updating each weight negatively proportionally to this gradient. Since the target is simply equation (4.5) and the actual output is just $\hat{V}(s)$, the full update rule can be written in vector notation as:

$$\Delta\mathbf{w} = \beta\big(r + \lambda\hat{V}(s') - \hat{V}(s)\big)\boldsymbol{\phi}(s) \tag{4.8}$$

Prescott considers the slightly more complicated case of $TD(\lambda)$ for arbitrary λ (the above account is for the special case of $\lambda = 0$), but the analysis is similar.

The actor network is constructed in the same way as the critic, with the input being recoded into $\boldsymbol{\phi}(s)$ and then the elements being combined linearly according to a new weight vector, $\mathbf{u}$. One difference is that the actor has two outputs which represent a mean, μ, and standard deviation, σ, of a single dimensional normal distribution. In a manner similar to Williams (1988), Gullapalli (1990) and Rummery (1995), the output of the actor is generated according to the probability distribution defined by these two parameters. For any action taken, the weight matrix[10] of the actor is updated (in a similar manner to the critic) to either increase or decrease the likelihood of producing that action again. If the perceived return (i.e. equation (4.5)) is greater than the expected return of the mean action μ (the output of the critic, $\hat{V}(s)$), then μ is moved towards the new action. If the perceived return is less than the expected return then the mean is moved away from the action. The standard deviation is updated too. If equation (4.5) is greater than $\hat{V}(s)$ then the current position of μ seems to be suboptimal and so σ is increased because more exploration appears to be useful. If $r + \lambda\hat{V}(s') < \hat{V}(s)$ then μ appears to be close to a local maximum in the action space, and so σ is reduced. In this way, exploration is controlled by how much the estimated returns of explored actions differ from the estimated expected return of the mean action. This is appealing because as μ gets closer to a local maximum, the exploration is automatically reduced, while if better actions are found away from μ then exploration is increased in order to facilitate movement to these better actions. A final point is that multi dimensional actions are easily achieved by using multiple and independent actors with their own sets of weights.

Prescott reports good results with this architecture, although refers to potential problems with breaking symmetries in the action space. For example, if the robot is approaching a wall head on, then on some occasions the agent will explore left turns and on others it will explore right turns. Both will be reinforced and so the actor will be drawn in two directions simultaneously resulting in a cancelling effect. This is a side effect of restricting each state to maintaining a single action. The new model introduced in chapter 5 will address this issue by allowing a parallel search of the action space which offers a choice of dynamic actions to each state. However, the work of Prescott remains interesting for its successful use of CMAC to generalise over a low dimensional state space[11], and for its application of dynamic real-valued actions to a delayed reward reinforcement learning problem.

[10]It is now a matrix because there are two outputs and therefore two sets of independent weights.

[11]CMAC suffers badly from the curse of dimensionality since the number of tiles increases exponentially with the dimensionality of the problem. This affect may be ameliorated to some extent by hashing.

CHAPTER 5

A New Model

5.1 Introduction

In the light of the account given so far, a number of desirable features for a candidate model of action space representation can be identified. Firstly, the representation of the action space should not be fixed beforehand but should be adaptable, and robust to non-stationary environments. The premise that it will not always be easy or appropriate to solve RL problems using a fixed set of actions renders a handcoding approach following the box pushing robot of Mahadevan and Connell (1991) unattractive. Although the coarse-coding algorithm reviewed in Santamaria et al. (1997) does provide dynamic generalisation, once the locations of prototypical Q-values are established in the combined state-action space, they are fixed. This drawback, in combination with a potentially involved search process for optimal actions, may lead to an inefficient representation.

A second desirable feature is that the standard RL theory should be closely adhered to with a central concept of estimating expected reward, full support for a real-valued and discounted reward signal, and an ability to represent real-valued states and actions. Departures from the theory which result in compromising generality include the approaches of Touzet's Kohonen mapping of the combined input-action-reward space, Ackley and Littman's CRBP algorithm, Touzet's agonist/antagonist backpropagation network and Ziemke's application of CRBP to robot learning. Particular shortcomings are noted as binary reward signals, binary representations of states and actions, no explicit maintenance of estimated expected reward (thus losing sight of the goal — namely to maximise this value), and no provision for discounted reward.

A third desirable property is for the values of *multiple actions* to be maintained for each state. This is a principle of Q-learning, for example, which maintains estimated expected reward for *all* possible state-action pairs. In contrast, the Motoric Map of Ritter et al. (1990) and its 'covariance' extension (Wedel and Polani, 1996) both maintain only a favourite action for each state. Being able to select an action from a list of alternatives has positive implications for robustness, particularly in situations where one or more action types may be inhibited. Maximising reuse of learned actions is also a key consideration, since we expect exploring the action space to be a costly process (as noted by Wedel and Polani (1996)). The Motoric Map approach fails to satisfy this property, since actions learned in one state may not be utilised by other states.

The findings of the review of the previous chapter are summarised with respect to the above criteria in the table of figure 5.1. This table should be interpreted as a guide for further discussion, rather than an authoritative last word on the success or failure of specific algorithms, and will be discussed in more detail in section 6.7. In the meantime we can use this summary both to guide the search for a new model, and to evaluate the success of any such model.

Since the type of problems of interest to this thesis are those in which no environment model is available, a final necessary property is that the learning mechanism is not model-based. With all these points in mind, as well as the more general issues of efficiency, simplicity, and scalability, a new model is now introduced. For the sake of clarity, the model is introduced incrementally. We will first address the issue of representing the state space, and then later extend the model to the action space, which is the main focus of the thesis. Although the actual approach for representing and

		Coarse-coding					SOM			Backpropagation					
		Handcoding (Mah. & Con.)	*Coarse-coding*	*RBF (Santamaria)*	*CMAC (Prescott)*	*Nearest Neighbour (Moore)*	*SOM (Touzet)*	*Motoric map (Ritter et al.)*	*Covariance learning (Wed. & Pol.)*	*CRBP (Ackley et al.)*	*CRBP (Ziemke)*	*QCON (Lin)*	*Backprop (Touzet)*	*SRV units (Gullapalli)*	*Q-AHC (Rummery)*
	Dynamic generalisation	✗	?	?	✓	?	✓	✓	✓	✓	✓	✗	✓	✓	✓
Generality	Adhere to RL theory	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✓	✗	✓	✓
	Real-valued, discounted reward	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✓	✗	✓	✓
	Real-valued states and actions	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓
Flexibility	Re-use of actions	✓	?	?	✗	?	?	✗	✗	✗	✗	✓	✗	✗	✗
	Multiple actions for each state	✓	✓	✓	✗	✓	✓	✗	✗	✗	✗	✓	✗	✗	✓
	Interpolation	✓	✓	✓	✗	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗
Scalability	Low update cost	✓	?	?	?	✓	✓	✓	?	✓	✓	✓	✓	✓	✓
	Low access cost	✓	✗	✗	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
	Low memory use	?	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓
	Scalable	?	?	?	?	?	✗	?	?	✗	✗	?	✗	?	?

Figure 5.1: Some existing approaches to action space generalisation are compared with respect to the identified set of desirable properties. The algorithms are grouped into non-neural, SOM-based, and backpropagation-based for convenience. A tick indicates that the algorithm satisfies the criterion, a cross that it does not, while a question mark represents uncertainty. It is emphasised that this table is not intended to be definitive, but rather to stimulate discussion on aspects of performance.

generalising over the input space is precedented by ideas introduced in the previous chapter, it will be convenient to start from the beginning since later extensions to the model will build on these foundations.

5.2 The simulator

For extra clarity, the model is introduced with respect to a specific problem. The problem is defined within the context of Olivier Michel's Khepera robot simulator (Michel, 1996) which is used as an experimental platform for many of the experiments of this chapter. The Khepera robot itself (Mondada et al., 1993) (as described briefly in section 4.8) is circular and has eight distance sensors, six on the front and two on the back. Only the six front sensors are used here. These sensors are short range (no more than the diameter of the robot, at least in the simulator) and act like feelers. Obstacles can be detected at very short range — just enough to give the robot time to take some evasive action if necessary. Each sensor reads a value between 0 (no obstacle in the line of this sensor) and 1023 (an obstacle is touching this sensor). These values are henceforth normalised to between 0 and 1 respectively and represent the input to the robot's control system. A sample environment corresponding to one square metre in the real world is shown in figure 5.2. The robot is shown approaching an obstacle, and the corresponding sensor readings can be seen in an enlargement of the robot in figure 5.3(a). Notice how the two central sensors record a higher activation than the peripheral sensors since the obstacle lies more centrally within their receptive fields (shown by the lines drawn from each sensor).

The input space can be visualised as the six-dimensional unit hypercube with each dimension corresponding to one of the sensors, and each sensory input vector represented by a point within this cube. The idea is shown in figure 5.4 and for clarity the sensors are labelled in figure 5.3(b). The numbering is intended to reflect the importance of the sensors, given that the tasks are going to involve moving generally forwards.

The outputs of the robot are two differential drive motors, one on either side of the vehicle, which can independently take values in the continuous range between -10 (reverse at full speed) and 10 (forwards at full speed). These values are henceforth normalised

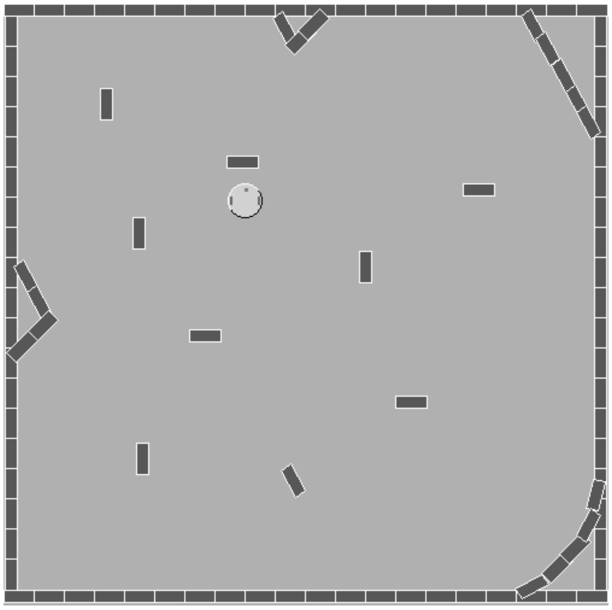
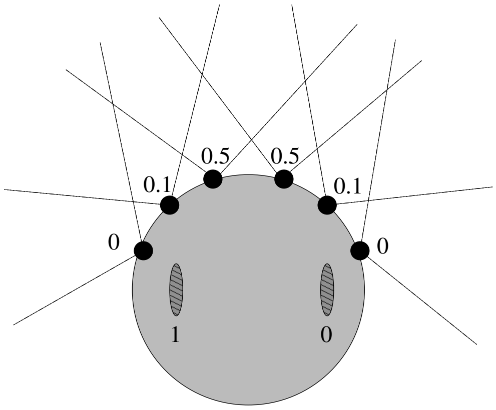
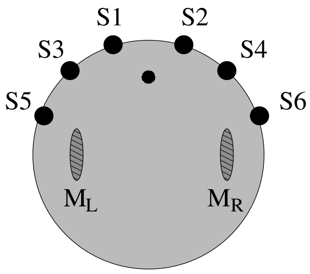


Figure 5.2: A sample simulated environment with simulated bricks forming obstacles and the environment boundary.



(a) An enlargement of the robot in figure 5.2 with the receptive fields of each sensor added.



(b) Sensor and motor labelling.

Figure 5.3: The robot

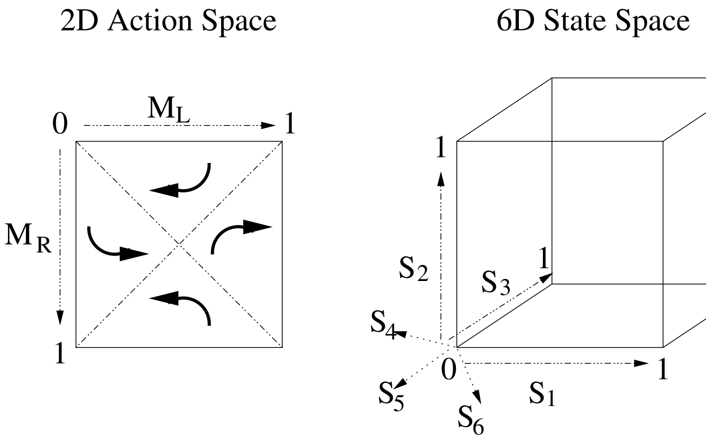


Figure 5.4: The six-dimensional sensor space and two-dimensional motor space. With respect to the motor space, the point in the middle of the space corresponds to the situation when both motors have a value of 0.5, i.e. when both motors are stationary and therefore so is the robot. The arrow in each quadrant is intended to denote the direction of travel given that the robot is facing the top of the page when an action within this quadrant is invoked. The motors are drawn on the robot diagrams, and figure 5.3(a) shows the robot involved in a right turn on the spot.

to between 0 and 1 respectively. By setting the wheel motors to appropriate speeds a wide range of different movements can be achieved encompassing both forward and reverse turns with $0 \leq \text{radius} \leq \infty$ and $0 \leq \text{speed} \leq \text{max. speed}$, as indicated in figure 4.8. Different combinations of motor values generate different directions of movement which are broadly classified in figure 5.4. This diagram represents the motor or action space as the unit square with any motor combination being represented by a point within this square. The top left corner corresponds to the robot travelling straight backwards at full speed, the bottom right corner to the robot travelling forwards in a straight line at full speed, the top right corner to a right turn on the spot at full speed, and the bottom left corner of the space to a left turn on the spot at full speed. The point in the middle of the space corresponds to the situation when both motors have a value of 0.5; i.e. when both motors are stationary and therefore so is the robot. The arrow in each quadrant is intended to denote the direction of travel given that the robot is facing the top of the page when an action within this quadrant is invoked. The motors are drawn on the robot diagrams, and figure 5.3(a) shows the robot involved in a right turn on the spot.

As a built-in feature of the simulator, uniform random noise of 10% is added to the amplitude of the motor speed and uniform random noise of 5% is added to the direction resulting from the difference of the speeds of the two motors. Uniform random noise of 10% is added to the distance calculated by the sensor readings. These noise factors help add realism to the simulator, although in the author's opinion there is already considerable noise inherent in the simulator as a result of inaccuracies in the calculation of the sensor readings.

5.3 The control system

The first task to be performed is one particularly familiar to the robot learning literature — obstacle avoidance (Tani and Fukumura (1994); Touzet (1997); Ziemke (1996)). The aim is for the robot to wander around the environment avoiding obstacles en-route. The agent is first given a default behaviour which moves the robot forward at full speed if there is no obstacle in sight. No obstacle in sight is inferred if each of the six distance sensors has an activation of less than 0.2. If at least one of the sensors does have an activation more than 0.2, then control is passed to the obstacle avoidance behaviour whose role it will be to take evasive action. Hence only a single behaviour need be considered, for simplicity.

Figure 5.5 illustrates the basic RL framework. A discrete set of states $\{u_1 \dots u_n\}$ represents all the situations the robot can find itself in, while a discrete set of actions $\{a_1 \dots a_3\}$ represents what the robot can do in each state. In this case (and in a similar manner to the handcoding of Mahadevan and Connell (1991)), there are three fixed actions corresponding to a right turn on the spot, forward at full speed, and a left turn on the spot. The normalised motor values are also shown and can be visualised occupying three corners of the action space of figure 5.4. For each state-action pair, represented by the connecting lines, an estimate of the expected discounted reward of taking that action in that state is maintained and updated during learning. For this reason these estimates are labelled *Q-values* after *Q-learning*, although the estimate here will be calculated according to a slightly simpler method — one which is more accurately considered a non-bootstrapping, Monte-Carlo method. At this point we are faced with the choice of either calling these values something different — say M-values — and henceforth continually pointing out the relationship between these values and

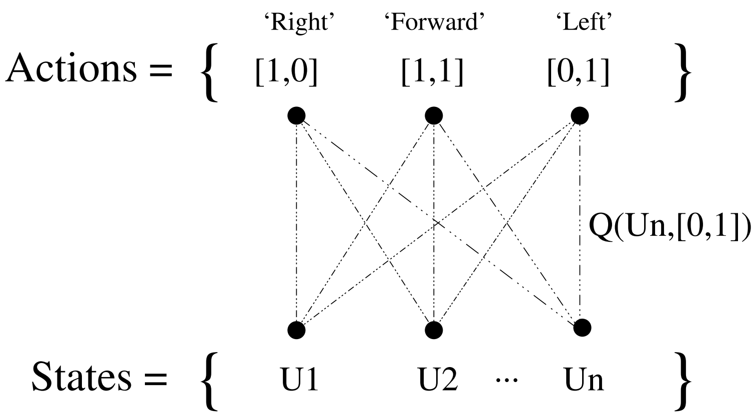


Figure 5.5: Basic RL framework.

Q-learning, or just importing the more familiar terminology with the caveat that the actual method of calculation is slightly simpler and is not considered *faithful* Q-learning. The value estimation method itself is not under scrutiny in this thesis, and indeed any non-model based method for estimating the expected reward of state-action pairs (be it Q-learning, $Q(\lambda)$, Monte Carlo etc.) could be substituted without loss of generality.

5.3.1 The reinforcement function

At each time-step the robot can make a movement corresponding to a fraction of its body length if it is travelling forward at full speed. The agent is therefore able to switch its behaviour quickly. Also at each time-step, the agent is perceived to be in one of the states, $\{u_1 \ldots u_n\}$, whereupon one of the three actions, $\{a_1 \ldots a_3\}$, is taken, eliciting a scalar reward from the environment. Let us say that at time t the agent is in state u_t, takes action a_t and receives reward r_t. Since we are only currently concerned with the obstacle avoidance behaviour, r_t is chosen as:

$$r_t = \begin{cases} 1 & \text{If no sensor activity is above 0.2} \\ -S1 - 0.5 \times S3 - 0.25 \times S5 & \\ -S2 - 0.5 \times S4 - 0.25 \times S6 & \text{Otherwise} \end{cases} \tag{5.1}$$

Hence the agent is given negative reinforcement proportional to activity on its distance sensors with a weighting appropriate to the significance (centrality) of those sensors.

5.3.2 The learning rule

The learning rule uses a fixed receding horizon, h, and the familiar discount factor, γ, and at time $t+h$ updates the estimate $Q(u_t, a_t)$ towards the discounted sum: $r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^h r_{t+h}$ by a learning rate, α. The actual update rule can then be written as:

$$Q(u_t, a_t) = Q(u_t, a_t) - \alpha \Big\{ Q(u_t, a_t) - (r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^h r_{t+h}) \Big\} \qquad (5.2)$$

which sufficiently resembles the expressions found in chapter 2 to convince us that the target is indeed the relevant part of the expected discounted reward, providing that the effect of taking any action in any state on the received reward will be delayed by at most h time-steps. The approach most closely resembles the Monte-Carlo technique (section 2.6) since Q-values are updated towards *actual* returns rather than other estimates. The reason this particular method is adopted is based on the assumption that the effect of any action will be restricted to a finite window of time following that action. So even though the length of each trial is effectively infinite, a truncated Monte-Carlo approach is still deemed appropriate. The degree to which bootstrapping estimates are desirable is still an open question in the literature (Sutton, 1996), although the likelihood is that an intermediate approach as offered by $Q(\lambda)$ will generally be prefered.

As the state-action pair (u_t, a_t) is visited again and again, and alpha is slowly reduced to zero, $Q(u_t, a_t)$ will converge on $E(\sum_{a=0}^{h} \gamma^a r_{t+a} | u_t, a_t)$ for the current policy (which we hope will converge to an optimal policy). The actual parameter values used are: $h = 4$, $\gamma = 0.95$, and $\alpha = f(t)$, with $f(t)$ slowly decayed from 1 to 0 during the course of the trial according to $f(t) = 0.9999^t$ (see the graph of figure 5.11, to come)[1].

[1]Note that a value of $h = 4$ means that only the reward received in the four time-steps immediately following an action can be credited to that action. For problems with longer delayed reward, a larger

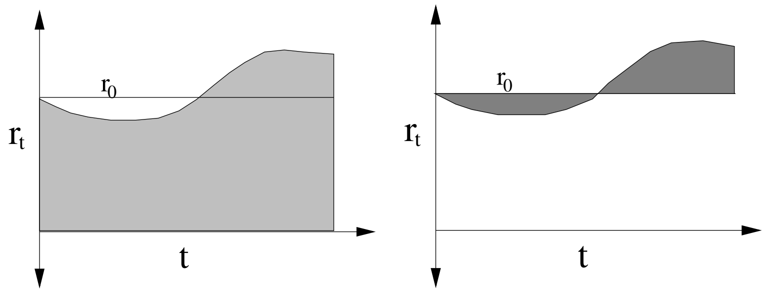


Figure 5.6: Two ways of summing reinforcement. If there is continuity in the reward signal, then estimating the area shaded in the graph on the right may allow faster learning. For the experiments that follow (unless otherwise stated), the approach on the right was empirically found to perform better and was therefore adopted. However, the standard approach (left) is generally preferred because it adheres to the theory.

Note that a policy, π, is being learned which maximises the following for all u_t.

$$E\left(r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^h r_{t+h} | a_t, u_t, \pi\right) \tag{5.3}$$

Figure 5.6 shows a sample set of reinforcement values varying over time. The first graph illustrates the learning rule described above, with the area under the graph representing the undiscounted sum of reward immediately following the action taken at $t = 0$. $Q(u_0, a_0)$ will then be updated toward the shaded area under this graph, even though the bulk of this value is made up of the area under the r_0 line which corresponds to the reinforcement that state u_0 'inherited'. This idea of inherited reward assumes that there is dependency between the reward of subsequent time-steps, which is justified by referring back to the reinforcement function of (5.1). For example, if there is sensor activity at some time, t, then there is also likely to be sensor activity at time $t+1$ since the robot cannot make large changes to its sensor readings in any one time-step.

Although this poses no theoretical problem to convergence, practical difficulties may

horizon or a bootstrapping technique would be required, with no loss of generality to the model. This value of 4 was empirically observed to be optimal in terms of maximising the final level of performance without compromising learning speed. Other values for h also yielded good results.

arise when, as a result of state generalisation (to be described), on the next occasion that state u_0 is visited, it coincides with a different r_0. Now $Q(u_0, a_0)$ is updated to the area under *this* graph, and the problem is that if the area shaded in the second graph of figure 5.6 is small compared with the rectangle under the r_0 line, then the important information in the reinforcement signal — namely what has happened *since* taking the action — will be swamped or saturated by the strength of the background signal. Following these observations, in the experiments that follow, the reward signal is interpreted as the area shaded in the second graph and not the area shaded in the first. This involves amending the update rule of equation (5.2) so that r_{t-1} is subtracted from each r term before being multiplied by the discount factor γ.

A modest increase in performance was observed using this approach in favour of the more standard approach of equation 5.2/figure 5.6(left). However, while the idea of subtracting 'inherited reward' does not fit neatly into the theoretical framework presented in chapter 2, neither is it deemed critical to the success of the model[2]. The details are included here for reproducibility rather than theoretical significance. In applications where there is little or no dependency between the reward values of subsequent time-steps, the standard approach would appear to be favoured. Indeed, the standard approach is recommended as the first approach to any problem.

5.3.3 Exploration

Following the simple learning rule, a very simple exploration strategy is adopted. At each time-step, the action with the highest Q-value for that state is chosen with probability $1-p$, and one of the other actions chosen at random with probability p. p is *annealed*[3] from 1 to 0 throughout each run in exactly the same way as the learning rate α — i.e. according to $f(t)$. This implements a time dependent transition from exploration in the early stages to exploitation in the latter stages. Linking the annealing schedules of α and p is largely arbitrary and mainly for convenience. Setting the

[2]Experiments (unreported) performed with the more standard learning rule of 5.2 also yielded good results.

[3]We borrow the terminology from the process of cooling metals slowly in order to toughen them. Here, the exploration (more generally the plasticity) fulfils the metaphoric role of temperature. We will often refer to the specific way in which learning rates are reduced over time as the *annealing schedule*.

schedules will be discussed in more detail later, but in the meantime we note that the learned policy using these (interlinked) parameters was empirically found to be as good as that obtained with any other (independent) parameters in terms of post-learning performance.

5.3.4 Mapping the input space

The next question to consider is how to arrive at a small and suitable set of states, $\{u_1 \ldots u_n\}$. The approach adopted here follows that of Touzet (1997) and the Motoric Map of Ritter et al. (1990), and uses a two-dimensional Kohonen map consisting of a 5×5 array of units to map and organise the input space. The inputs to the network are the six distance sensor values of the robot, with a new input being received at each new time-step. Now each unit of the network can simply be used as one of the states in figure 5.5 resulting in the basic architecture shown in figure 5.7. Note that the Kohonen network is mapping the sensory data, and is not producing a physical map of the environment.

An empirical observation is that a Kohonen map is most successful when inputs are evenly distributed during the formation of the map. However, because the inputs will be generated by the robot's physical interaction with a continuous environment, and because there will therefore tend to be dependencies between these stimuli, we expect the units of the map to be pushed and pulled around in the weight space unevenly. To address this, input stimuli are stored in an overwriting buffer with room for 100 previous sensory experiences. At each time-step one of these stored stimuli is chosen at random (uniformly) and used to update the network, rather than the current sensory information being used. Buffering the input has precedents in the literature — Tani et al. (1997) for example do this in a different setting but for similar reasons.

The flow of control is now summarised. At time t:

❶ Present input stimulus, I_t, to the Kohonen network.

❷ Identify winning unit, u_t (with smallest Euclidean distance from I_t).

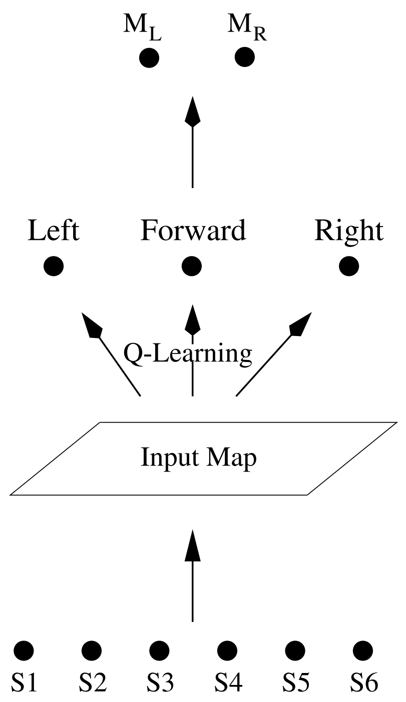


Figure 5.7: The basic architecture. A SOM maps the six dimensional input space (corresponding to the six sensors, $S1 \ldots S6$) with each unit in the map representing a distinct state of the standard RL problem (see figure 5.5). The activation of the six input units can be directly interpreted as the normalised activation of each sensor.

❸

$$\text{Take action, } a_t = \begin{cases} \text{One with best Q-value} & \text{If } \textbf{Exploiting} \\ \text{Random action} & \text{If } \textbf{Exploring} \end{cases}$$

❹ Run simulated robot on a_t, generating a reward, r_t, and a new input vector, I_{t+1}.

❺ Calculate the discounted reward of the state-action pair of h time-steps ago now that all the reward is in for that pair:[4]

$$R = r_{t-h} + \gamma r_{t-h+1} + \gamma^2 r_{t-h+2} + \ldots + \gamma^h r_t \tag{5.4}$$

[4]This implements the basic return shown in the left plot of figure 5.6. The relative return shown in the right plot of figure 5.6 is easily achieved by subtracting r_{t-h-1} from each reward term before multiplying by the factor, γ.

❻ Update $Q(u_{t-h}, a_{t-h})$ towards R by learning rate α.

❼ Update the SOM using an input vector picked at random from a short term buffer.

❽ Return to ❶.

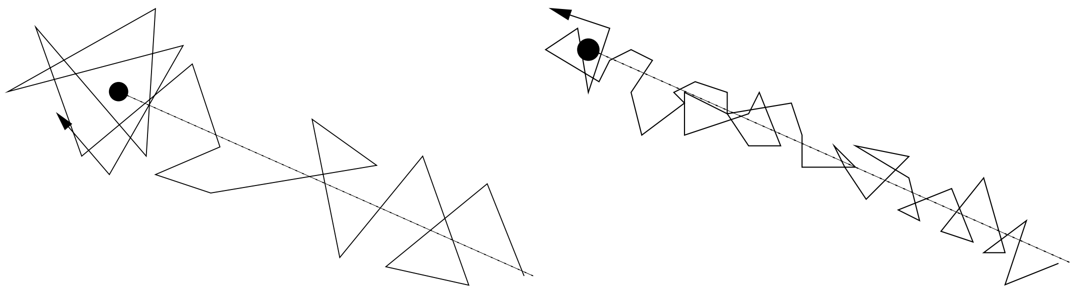
This cycle takes place only when the obstacle avoidance behaviour is active; i.e. when there is significant sensor activity. The exceptions are that the Kohonen map may still continue learning using its buffered stimuli, and that there may be some state-action pairs still waiting to collect reinforcement in the h time-steps following their invocation.

So learning proceeds in parallel in the Kohonen map (henceforth called the *Input map*) and on the Q-value connections between each unit of the Input map (henceforth referred to as *Input units*[5]) and the three actions. As the trial proceeds, the probability of exploring, the Q-learning rate, the learning rate of the Input map and the neighbourhood of the Input map are all annealed according to the annealing function $f(t)$. The exact nature of the annealing function is illustrated shortly, but the motivating principle is ubiquitous in incremental stochastic systems. A high learning rate results in fast learning that is sensitive to noise while a low learning rate will slow down learning, but increase robustness to noise. An analogy is that the most efficient way to fill a cup from a tap is to start with high pressure, and slowly turn the tap off as the water level reaches the top. Figure 5.8 illustrates the concept.

5.4 Results

There are a number of ways to visualise this type of architecture at work. The most obvious visualisation is a snapshot of the agent wandering around its environment and this is shown in figure 5.10. The trajectory shows the agent takes left turns when obstacles are encountered on the right, and right turns when the obstacle appears on the left. This suggests that the learning system has done a good job of optimising reward,

[5]A capital letter will be used for 'Input map' and 'Input unit' in order to distinguish the named parts of this particular architecture from more general references to maps, inputs and units.



(a) A large learning rate yields fast learning but is sensitive to noise which may prevent the target being reached accurately.

(b) A smaller learning rate is more robust to noise but progress towards a target may be slow.

Figure 5.8: An abstract illustration of the effect of different learning rates.

and also, importantly, that the reward signal of equation 5.1 is both appropriate and effective with respect to the task. This is not always obviously so. The trajectory plotted in figure 5.10 is after learning has taken place. At the beginning of the run, the robot's obstacle collision behaviour is virtually random. A sample of this is shown in figure 5.9.

The graph of figure 5.11 shows more quantitative results. The figure shows two increasing plots of average reward against the number of time-steps. The solid line, A, is the result of using the solid exponentially decreasing annealing function $f(t)$, and the dashed line, B, the result of using the faster dashed annealing function (0.9997^t). The top horizontal line shows the performance of the handcrafted (non-learned) obstacle avoidance strategy that was empirically judged to yield highest reward:

$$\text{Action} = \begin{cases} \text{Turn Right} & \text{If } (S1 + 0.5 \times S3 + 0.25 \times S5) > (S2 + 0.5 \times S4 + 0.25 \times S6) \\ \text{Turn Left} & \text{Otherwise} \end{cases} \tag{5.5}$$

The agent receives a default reward of 1 while it is performing its default forward

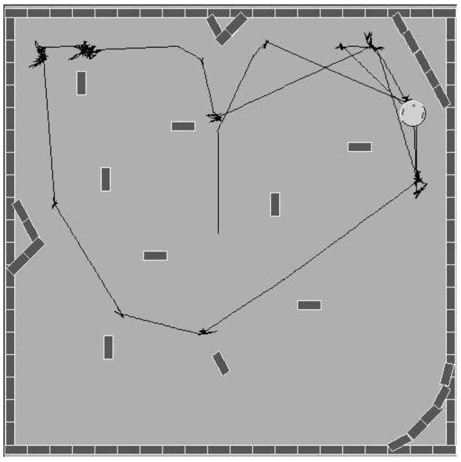


Figure 5.9: A sample path of the robot during a single trial, before learning takes place. The obstacle avoidance behaviour is significantly sub-optimal.

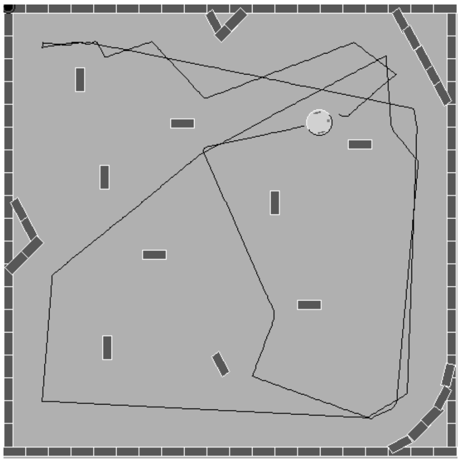


Figure 5.10: A sample path of the robot during a single trial, after learning has taken place. The trajectory represents about 1000 time-steps. Obstacles are successfully avoided (at the limit of the short-range sensors).

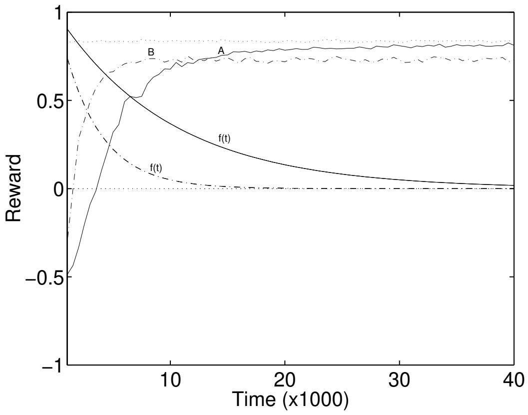


Figure 5.11: Graphs of reward against time with annealing functions. The top dotted line shows the performance achieved with the handcoded policy of (5.5). Curve A shows the average reward for the solid annealing schedule, while curve B shows the average reward for the faster annealing schedule indicated by the dashed line.

behaviour as this is considered a good state to be in, and therefore 1 is the theoretical maximum reward. But since the agent must spend some of its time avoiding obstacles, the practical maximum is less than this. The minimum reward per time-step can be calculated as -3.5, and would be received only when every sensor is saturated with activation.

The graphs in figure 5.11 are averaged over twenty runs. Because of the stochastic nature of the environment, the robot will find itself in different situations at the same point in time in different runs, and it is therefore necessary to average over the environment to achieve an accurate and consistent measure of performance. In fact, results are calculated only every 1000 time steps, so each unit of the x-axis is actually the result of averaging approximately $1000 \times 20 = 20000$ individual reward values, and so is a good indicator of the expected reward at that time step. Variances are discussed as part of the next experiment.

The main factor which affects the shape of the performance graph is the exploration parameter. The faster exploration is reduced, the faster performance increases because exploration represents noise to the system. However, as the dashed lines of figure 5.11 illustrate, if exploration is reduced too quickly, an optimal strategy may not be discovered since learning is also fuelled by exploration. The average reward at any given time-step is strongly related to the average amount of exploration at that time step. Therefore to judge the efficiency of the learning mechanism, it is necessary to discover how quickly the exploration may be annealed without sacrificing optimal behaviour[6] at the end of the run. In fact the solid lines represent this optimal annealing rate for this experiment, and unless otherwise stated all subsequent graphs represent optimal annealing too.

For clarity, the parameters are detailed below:

[6]By *optimal behaviour* we mean the highest reward observed for *any* annealing schedule.

Parameter	Value
Probability of Exploration, p	$f(t)$
Q-Learning Rate, α	$f(t)$
Learning rate of Input map, IL	$0.3 \times f(t)$
Input map neighbourhood size, IN	$5 \times f(t)$
$f(t) = c^t$, for various $c < 1$	Optimal $c = 0.9999$

It turns out to be both convenient and empirically sound to anneal all these parameters with the same function, $f(t)$[7], although the starting values are sometimes usefully varied (for example IL and IN), and some of the parameters may be given a minimum value. For example, the Input map may form more evenly if a small minimum neighbourhood is always maintained (see appendix A for notes on the exact calculation of the neighbourhood). It has already been stated that linking all annealing schedules to $f(t)$ is somewhat arbitrary. In theory, and in practice, a better strategy may involve different schedules, but a complete analysis of the parameter space is considered beyond the scope of this thesis. The important point is that all values are annealed with respect to each other, so that no part of the system is left behind the others. In some cases different annealing schedules were tried, but none were able to improve on the performance of the proposed set of parameters.

The starting conditions of 5 for the Input map neighbourhood and 0.3 for the Input map learning rate are arbitrary and were empirically derived, although providing sensible values were selected, performance did not significantly vary. All weights of the Input map were initialised to random values in the range $[0,1]$ and all Q-values were initially set to a large negative value. This was to ensure that the exploration was driven by the exploration parameter (as is necessary in the general case), and not by coincidentally favourable initial conditions. Many systems of this kind also include some kind of emergency reflex in order to avoid becoming stuck (Li, 1999; Tani and Fukumura, 1994). Here a frustration counter is kept that increases with an absence of forward movement and decreases in its presence. If the counter exceeds a threshold (signifying

[7] In theory, $f(t)$ should satisfy the constraints of equation 2.14. In practice, since the trials are not of infinite length, disregarding these constraints should not significantly impact learning. However, care is required that $f(t)$ is not annealed too quickly, otherwise units and Q-values can become 'stranded'. The issue of parameters is addressed in chapter 6.

the agent may be stuck) increasing amounts of (exploratory) random noise are added to the motors (a similar idea is found in Lin (1991)). This reflex is not invoked often, but avoids degenerate behaviour. It is also noted that both sensory-motor noise and random action exploration may also help reduce problems associated with repetitive and degenerate behaviour.

5.4.1 The Input map

Figure 5.12 shows the Input map of one of the runs after learning. For each unit in the 5x5 array, six vertical bars are drawn corresponding to the six element weight vector of that unit in the order the sensors appear on the robot, from left to right. The shading of these bars corresponds to the value of that weight with black indicating a weight of 1 and white indicating a weight of 0. The circles in the top right corner of each unit are coloured according to the action for which that unit has the highest Q-value; i.e. the action it will take in the absence of exploration. White represents a left turn, black a right turn, and grey corresponds to the action for moving forward (which is never utilised). This type of plot is used extensively throughout this document.

The first observation is that an intuitively optimal mapping is achieved with left turns being taken when there is more activation on the right sensors than on the left, and vice-versa. A second observation is that physically neighbouring units respond preferentially to similar prototypical inputs. A third feature is that the two actions meet at the more ambiguous categories; note how row three contains units responding to states that only just warrant a left turn, whereas row five contains more extreme situations. The larger number of white circles indicate that the agent may have been involved in more left than right turns. Such asymmetries were indeed observed, although it is also noted that the SOM lacks any kind of equiprobable unit distribution guarantee, and so an equal distribution of units would not necessarily be expected even if the robot did encounter the two classes of situation with equal frequency.

Also illuminating is figure 5.13 which shows the same Input map displayed in weight space rather than the topographic space of the map. Figure (a) shows each unit plotted in the dimensions of the front two sensors, S1 and S2. Figure (b) shows the units in

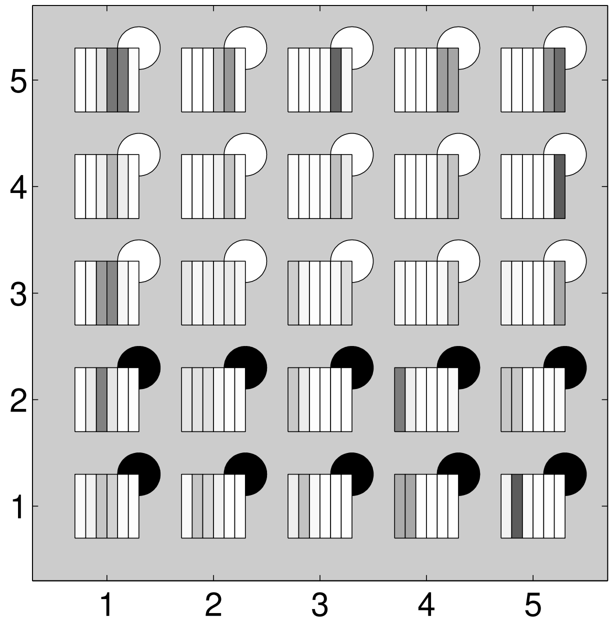


Figure 5.12: Diagram showing the 5x5 Input map. Each unit is represented by six vertical bars and a circle. The bars represent the weight vector of the unit and the circles (white = left turn, black = right turn) represent the action with the highest Q-value at that state.

the dimensions of the next two sensors, S3 and S4, and (c) shows the same set of units in the peripheral sensor dimensions, S5 and S6. Neighbouring units are connected by lines and the shading of each unit corresponds to its usage relative to the others with a dark unit indicating high usage. Again the topological preservation is visible, particularly in (a) and (c). It can also be seen how the units occupy only the active regions of the input space. For example, the agent rarely encounters more than 50% activity on any sensor because it learns to take evasive action before this point. Most units have ended up at the origin of the first map and on the axes of the second and third maps, from which we may deduce the agent tends to encounter obstacles from one side or another but rarely head on, a feature again validated by observations. However, it is also apparent from these diagrams that usage is by no means uniform, with some units winning the competition for control of the robot far more than others. Drawbacks associated with the non-equiprobable distribution of the SOM's units are well documented (Kohonen, 1995; Bishop et al., 1998; Hertz et al., 1994; Hulle, 2000).

Figures (d), (e) and (f) are identical to (a), (b) and (c) except that each unit is coloured

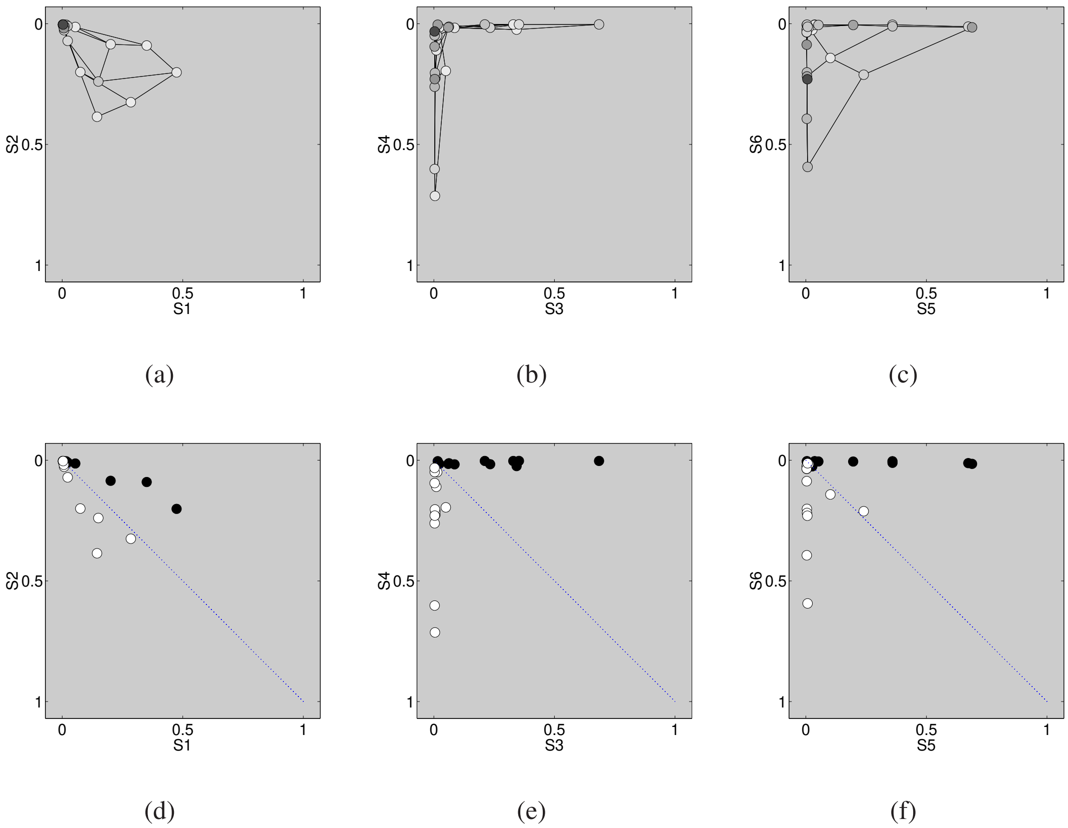


Figure 5.13: Visualisation of the Input map in weight space after learning. Each subfigure shows the Input map in two dimensions (corresponding to two opposing sensors) at a time (recall figure 5.3).

according to the action for which it has the highest Q-value according to the same scheme as before. The region above the diagonal line corresponds to situations with more activity on the left sensor of that pair than the right, and right turns are typically prescribed for units occupying this region. The converse is true below the line.

5.5 Utilising the topology of the Input map

Much of the effort expended in the field of RL is directed towards finding ways to use the reward information efficiently and distribute it to relevant parts of the value function as quickly as possible. Reward information must be propagated quickly to the states that were implicated in the reward's production, and in the case of temporal difference methods, to *their* typical predecessor states, or state-action pairs. Approaches include eligibility traces (Watkins, 1989), clustering of states with a Hamming distance measure (Mahadevan and Connell, 1991), and various methods for learning and utilising an environment model such as DYNA (Sutton, 1990).

Continuing in this spirit, and having already reduced the number of states to be considered using a fixed size SOM, the topological preserving feature of the network may now be exploited further. Since neighbouring units represent similar perceptual categories, it is convenient and appropriate for neighbours to learn from each other's Q-value updates. If this neighbourhood is annealed in the same way as before, then early in each trial all units get quick access to crude estimates, but by the end of the trial each unit must be responsible for the bulk of its own fine-tuning. The improvement in the learning speed when reward information is made available across the network in this way is illustrated in the graph of figure 5.14. Plot A is that of figure 5.11, i.e. the best that was possible with each unit learning only from its own experiences. Plot B shows the speedup in learning made possible by units being able to learn from their neighbours' experiences. The optimal annealing functions used in obtaining the two graphs are also shown as dotted lines, with the steeper curve representing the annealing function used to achieve graph B. The learning rule is identical to that of equation 5.4 except that now in step ❻ *every* unit is updated towards R proportional to the neighbourhood function, ψ.

Representing the input space using an SOM is a simple and effective approach that yields a concise set of relevant states for use with standard RL techniques. As an illustration, if the input space was divided up by hand into tiles of width 0.1 (for example), this would result in 10^6 states with many units being wasted, and yet still a lack of resolution in the more densely populated regions (leading to over-generalisation). Although not reported here, experiments performed with the state space handcoded in this way showed poorer performance. Further comparisons with other dynamic methods for representing the state space are not offered since the Input map is only being used as a means to an end — namely that of generating an efficient representation of the space that can be used in the following sections[8].

[8]Although a number of alternatives are briefly considered as part of the analysis of chapter 6.

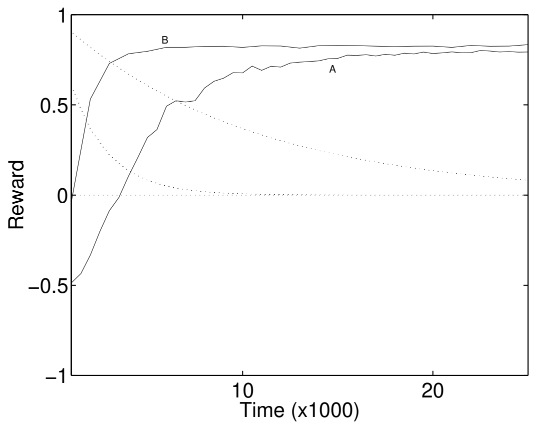


Figure 5.14: Using topology preservation to speed up learning. Graph B is with topology learning, graph A without. As before, optimal annealing rates are indicated by the dotted curves. The graphs do not start at the same point because the first data point is the result of averaging over the first 1000 time-steps during which the topological learning has already had a significant effect.

5.6 Learning actions

The architecture is now extended to consider the dynamic exploration, representation and generalisation of the action space. The approach is based on a second Kohonen network, which will henceforth be called the *Motor map* or *Output map*, with each unit of this map representing an action. In the following experiments a one dimensional Motor map is used, so the set of actions can be visualised as a line of units inhabiting the two dimensional output space of figure 5.4. The Motor units start at random positions within this space and learning proceeds as before, except that now an additional goal is to move these units to regions of the action space which are appropriate for maximising the expected discounted reward of each and every Input unit.

Assume that at time t, unit u_t of the Input map is the winner for the current stimulus and unit a_t is the Motor unit or Action unit which is selected in the usual way; i.e. the unit which maximises $Q(u_t, a_t)$ with probability $1 - p$, and a unit selected at random the rest of the time. The position of unit a_t in the action space is determined by the weights, $\langle w_1, w_2 \rangle$, of that unit. Now random noise is added independently to these values, so that the actual action taken is:

$$\text{Left Motor} = w_1 + MA \times \text{random(-1,1)}$$
$$\text{Right Motor} = w_2 + MA \times \text{random(-1,1)}$$

where random(-1,1) denotes a random number drawn evenly from the continuous range $[-1,1]$, and where MA ('Motor Anneal') is reduced from 1 to 0 according to $f(t)$ in the usual way. Hence the actual weights of the winning Motor unit are used as a mean action, but regions of action space are continually explored within an ever decreasing radius around this mean. Let us call the perturbed action $\langle M_L, M_R \rangle$ with M_L the value used to set the left motor and M_R the value used to set the right motor. Over the next h time-steps the actual receding horizon return, R, of the state-action pair (u_t, a_t) (where the actual action taken by unit $a_t = \langle M_L, M_R \rangle$) is calculated in the usual way following equation 5.4. Also as before, the Q-value associated with this pairing of units is updated towards R. However, before this last step is performed, R is compared with $Q(u_t, a_t)$. If $R \leqslant Q(u_t, a_t)$, and assuming $Q(u_t, a_t)$ is an accurate estimate, then the perturbed action $\langle M_L, M_R \rangle$ appears to be no better than the existing action prescribed by $\langle w_1, w_2 \rangle$, and so the Motor map is not changed. However, if $R > Q(u_t, a_t)$ then $\langle M_L, M_R \rangle$ appears to be better than the action currently prescribed by the weights of a_t, and the Motor map is therefore moved towards $\langle M_L, M_R \rangle$ according to the usual Kohonen update rule. This will result not only in unit a_t modifying its weights, but also its physical neighbours moving through weight space according to the neighbourhood function. As the learning rate is made smaller and smaller, actions that statistically yield high reward for at least one Input unit should emerge.

Because of topology preservation, neighbouring units of the Motor map will tend to prescribe similar actions. Therefore, as with the Input map, the learning of Q-values can again be speeded up by allowing neighbours to learn from each others' Q-value updates. So each time a Q-value is updated, *every* pair, $Q(u_n, a_m)$ is updated toward that same value by a factor that decays with the product of the distance of u_n from the winning unit of the Input map, and the distance of a_m from the active unit of the Motor map. If ψ and ψ_1 are the neighbourhood functions of the Input map and Motor map respectively for the pair (u_{t-h}, a_{t-h}), then at time t, $Q(u_{t-h}, a_{t-h})$ is updated toward the

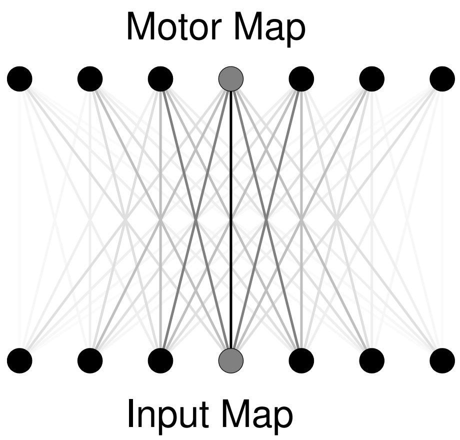


Figure 5.15: Q-values are updated proportionally to the product of the neighbourhoods of the relevant Input and Motor units.

discounted reward R (of equation 5.4) by a factor of $\alpha \times \psi \times \psi_1$. Figure 5.15 illustrates the two maps (although the Input map is one dimensional in this case), with the two active units highlighted, and the degree to which each connection is updated towards R also shown (the bolder the connection, the larger the update). Intuitively, the further a Q-value from the *actual* state-action pair that is being updated, the less it is updated itself, because the less likely the reward information is to be relevant. At the beginning of learning, when the neighbourhoods are large, any reward experience is considered better than none, but at the end, when the neighbourhoods are small, each state-action pair must essentially learn for itself.

For clarity, the basic algorithm for time t is outlined below:

❶ Present input stimulus, I_t, to the Input map.

❷ Identify winning unit, u_t.

❸

$$\text{Motor Unit, } a_t = \begin{cases} \text{One with best Q-value for this state} & \text{With probability } 1-p \\ \text{Random action} & \text{With probability } p \end{cases}$$

❹ Take action, $\langle M_L, M_R \rangle = \langle w_1 + MA \times \text{random(-1,1)}, w_2 + MA \times \text{random(-1,1)} \rangle$

where $\langle w_1, w_2 \rangle$ are the weights of unit a_t.

❺ Run the simulated robot on $\langle M_L, M_R \rangle$, generating a reward, r_t, and a new input vector, I_{t+1}.

❻ Calculate the discounted reward of the state-action pair of h time-steps ago now that all the reward is in for that pair:

$$R = r_{t-h} + \gamma r_{t-h+1} + \gamma^2 r_{t-h+2} + \ldots + \gamma^h r_t \tag{5.6}$$

❼ If $R > Q(u_{t-h}, a_{t-h})$ then update the Motor map towards $\langle M_L, M_R \rangle$ of h time-steps ago, proportionally to the Motor neighbourhood function, ψ_1, and Motor map learning rate, OL[9].

❽ Update the Input map towards an input vector picked at random from a short term buffer, proportionally to the Input neighbourhood function, ψ, and learning rate, IL.

❾ Update the Q-values of *all* state-action pairs towards R proportionally to $\alpha \times \psi \times \psi_1$, where ψ and $\psi 1$ are the neighbourhood functions of Input unit u_{t-h} and Motor unit a_{t-h} respectively.

❿ Return to ❶.

The idea behind annealing the radius of the exploration around each Motor unit is that at the beginning of the adaptation process the system must be able to search everywhere within the motor space, but towards the end of the run such exploration would be exceedingly costly, and at this point only ‘fine tuning’ exploration is appropriate. However, always maintaining *some* long range exploration may help the map find appropriate actions, even after MA is annealed. The problem is essentially one of local minima on the reward surface of the action space. For example, with respect to an obstacle avoidance behaviour, a unit cannot slowly move from the top-right corner of the

[9]The parameter OL stands for ‘Output Learn’ and is the learning rate for the Kohonen network that comprises the Motor map. As usual, it is linked to $f(t)$, just like its counterpart, IL.

action space (corresponding to a right turn) to the bottom-left corner (a left turn) just by following the reward gradient because the space in between (corresponding to moving forward with varying speeds) will typically yield much lower reward than either of the extreme turning actions. To ameliorate the general problem of local minima, noise is added to the motor weights according to the augmented schema:

$$\langle M_L, M_R \rangle = \begin{cases} \langle w_1 + MA \times \text{random(-1,1)}, w_2 + MA \times \text{random(-1,1)} \rangle & \text{with prob. } 1-q \\ \langle w_1 + \text{random(-1,1)}, w_2 + \text{random(-1,1)} \rangle & \text{With prob. } q \end{cases}$$

where q is annealed from 1 to 0 in the usual way with $q = p = f(t)$, and where the winning action unit is selected in the usual way. The exact impact of this particular measure is inconclusive, and it is not even clear that it is necessary at all. It is mentioned here because out of all the different techniques that were toyed with during the development of the algorithm, this one often seemed to find its way into final versions of experiments. It is included here for the sake of strict reproducibility rather than practical significance, although see the discussion on local reward minima in section D.5 for a possible use.

There are a number of other peripheral design details which are not considered essential to the algorithm, but which may be relevant for reproducibility. The foregoing experiment was performed at an early stage when many of these 'peripheral issues' were still being investigated as potentially significant. One such detail is that after an action has been perturbed, a new Action unit is selected which most closely represents the perturbed action, and it is *this* second Action unit which is deemed responsible for the behaviour. Obviously as the exploration parameter, *MA*, becomes smaller, this issue becomes less and less relevant. This and other peripheral details are described in appendix B for completeness.

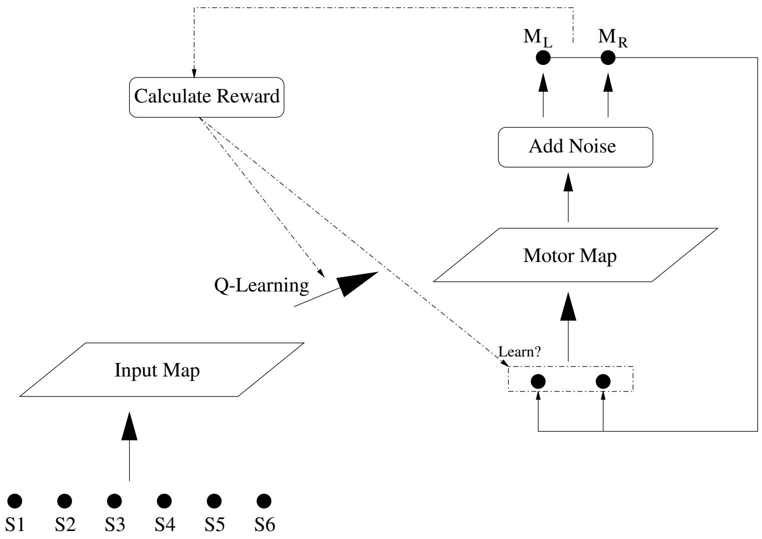


Figure 5.16: Basic Architecture.

5.6.1 Design overview

The algorithm is reviewed in figure 5.16. The system consists of two Kohonen networks, one forming a map of the input space and the other forming a map of the output space. The units in the Input map constitute states and the units in the Output map represent actions. For each state-action pair, a Q-value is maintained which is the estimated expected discounted reward of taking that action in that state. The Output map is slightly unusual in that it is trained towards random perturbations of its own units' weights, providing these perturbed actions yield suitably high reward. This is how new actions are generated.

One point worth re-iterating is that the flow of control through the system is separate from learning. Flow of control begins with the current sensory input being presented to the Input map, the winner being used to select a Motor unit, the weights of which are then used as a basis for the output. Independently and concurrently, the Input map is learning from its buffer of stored stimuli, the Motor map is learning towards those of its perturbed outputs which yield suitably high reward, and the Q-learning takes place in the usual way on the state-action connections between the two.

Adaptation is largely encapsulated, with the Input map forming without reference to the rest of the system and the Motor map forming with reference only to the Q-values and the reward signal. Within each map, units compete with their neighbours for activity, and updates are made locally without reference to other parts of the system. However, there is significant communication between all parts of the system through the environment. For example, the Input map formation affects the Q-value function which in turn influences how the Motor map forms. In a circular fashion, the Motor units control which actions are available and this alters the situations the agent finds itself in and hence the input to the Input map.

The role of competition

Input units will preferentially select certain Motor units depending on which ones yield highest reward. Therefore actions that can provide high reward to Input units will tend to be selected, and hence the rest of the Motor map will tend to be reinforced toward these actions, or at least to perturbed versions of these actions. This generates competition amongst high reward regions of motor space for Motor units.

It is possible that many Input units could favour the same Motor unit. This Motor unit will then attempt to optimise its position in action space with respect to the reward, despite the fact that it may be getting invoked in different situations requiring different optimal actions. In this situation each Input unit effectively casts a vote for where this Motor unit ought to be every time that state-action pair is invoked. If there are Input units that do not prefer the majority verdict, then they will tend to look elsewhere for Motor units they can control more effectively. However, all these features are only statistical tendencies, driven by random exploration, and mediated by a potentially noisy reward signal.

5.6.2 Results

The ability of the full system to learn appropriate actions within the continuous action space of the robot is now tested. However, because the agent can now choose from the entire range of possible forward and reverse directions of motion, and because moving backwards is the best short term strategy for improving the reward signal when presented with an obstacle, oscillating forward-reverse behaviour turned out to be an initial problem. The robot tended to back away from an obstacle until it could no longer perceive it, and then proceed with the default forward behaviour in a repetitive fashion. To address this, a graded punishment is added to the reward calculation of equation 5.1 to discourage actions in which the aggregate movement is backwards:

$$r_t = r_t - 2 \times (M_L + M_R - 1) \text{ if } M_L + M_R < 1 \tag{5.7}$$

The factor of two is to balance the strength of this punishment against the rest of the reward signal (see equation 5.1). Figure 5.17 shows the result of learning actions compared with the original experiment involving three fixed actions. Learning the actions takes considerably longer than when the actions are fixed, but near optimal behaviour is still discovered. The annealing function needs to be slower to allow for the greater number of alternatives to be explored, and this function is also shown on the plot as $f(t)$. For completeness, a summary of the parameters is also provided:

Parameter	**Value**
Input map size	5×5 units
Motor map size	20×1 units
Input map neighbourhood size, *IN*	$5 \times f(t)$
Motor map neighbourhood size, *ON*	$5 \times f(t)$
Reward horizon, h	4
Discount factor, γ	0.95
Q-learning rate, α	$f(t)$
Learning rate of Input map, *IL*	$0.3 \times f(t)$
Learning rate of Motor map, *OL*	$f(t)$
Probability of Q-learning Exploration, p	$f(t)$
Probability of Large Motor Exploration, q	$f(t)$
Max. Exploration distance around Motor unit, *MA*	$f(t)$
Annealing schedule, $f(t)$	0.9997^t

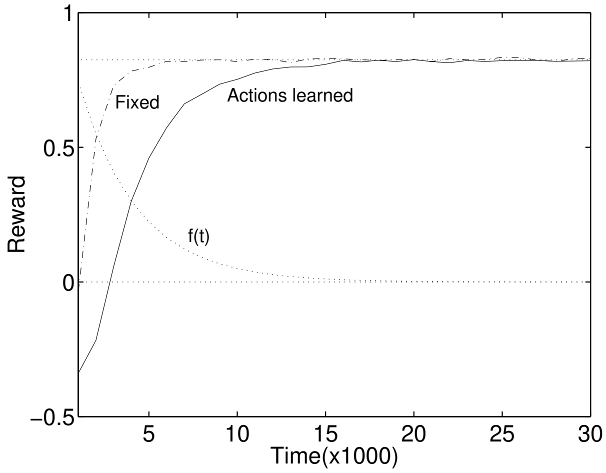


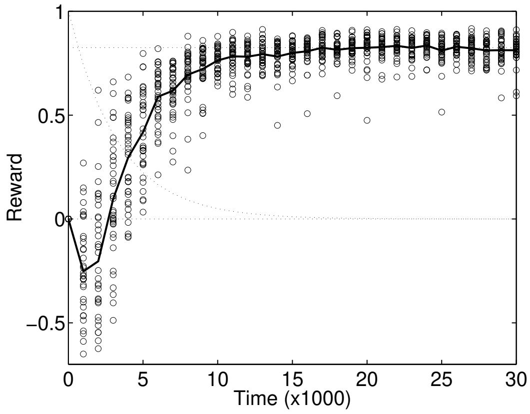
Figure 5.17: A direct comparison of the best results of learning actions compared with using three fixed actions (latter duplicated from figure 5.14). Annealing function for the former is also shown and the top dotted line is the handcoded strategy of (5.5).

In addition, some parameters had minimum values. The Input and Motor neighbourhoods had a minimum size of three to help form smoother maps, and the learning parameters also maintained small values to allow continued refinement of the maps throughout. A small amount of random noise was included in processes such as selecting the winning units of the maps, again to aid the formation of these maps[10].

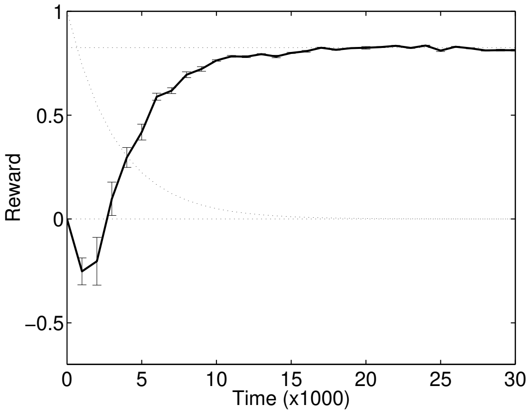
For completeness, the data from each of 36 independent trials is shown in figure 5.18(a). This graph provides a guide to the variability between runs. The actual variance (figure 5.18(b)) shows that this variability is greater at the beginning of each trial where the stochastic influences (and in particular the exploration) is greatest. This initial period represents a critical phase of learning during which all alternatives and all regions of the action space are explored. The standard deviation of the data is shown in (c) (simply the square root of (b)), and (d) shows the standard deviation of the mean. This indicates how the mean graphs of different sets of 36 independent trials would be expected to vary, and therefore how much confidence can be placed in any one such graph.

These graphs are considered typical for the kind of experiments performed as part of

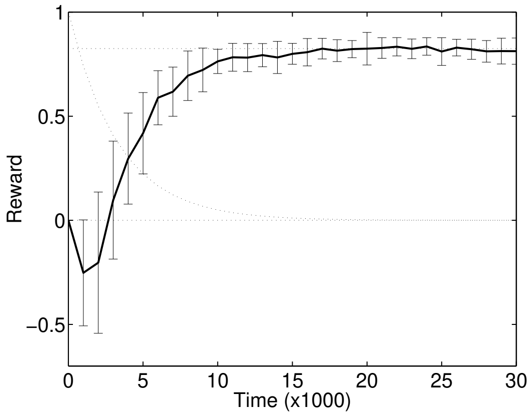
[10]Random and evenly generated noise in the range [-5%,+5%] was added to each weight comparison.



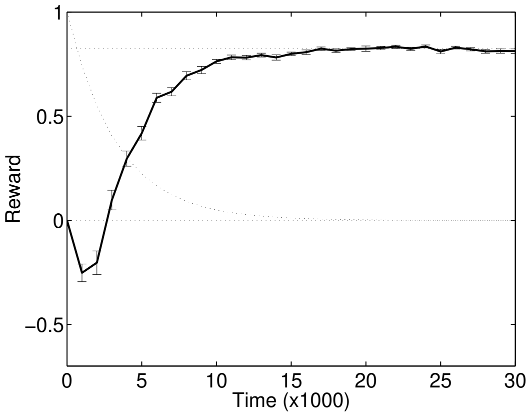
(a) The actual data of 36 independent trials. Recall that each data point is averaged over 1000 consecutive time-steps in order to average over stochastic environmental effects.



(b) The variance of the data shown in figure (a). Variance is greater at the beginning of each trial when learning is in its most dynamic phase. This is a critical phase since this is when the majority of the exploration takes place.



(c) The standard deviation of the data shown in figure (a). Assuming the data is generated according to a Normal distribution (clearly not true because of the asymmetry in (a)), then we can predict that approximately five in six data points will lie within the bars shown.



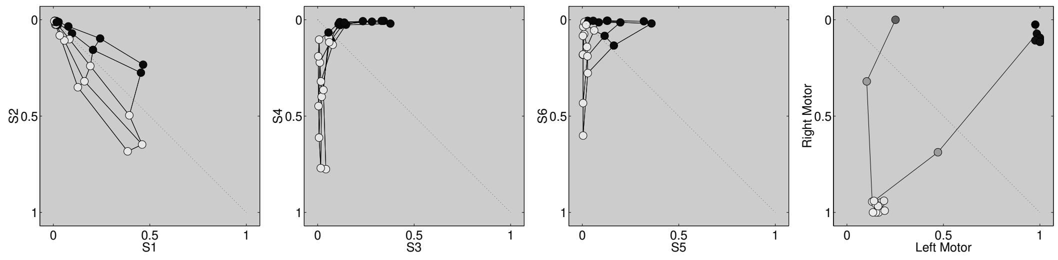
(d) The standard deviation of the mean of the data in (a). This graph indicates the expected variance of the main graph of figure 5.17 that would be generated from multiple data sets of this size.

Figure 5.18: Considering the variance and standard deviations of a typical set of runs. This time the x-axis is started at $t = 0$ (for which the reward is undefined, not zero as shown), so the first well defined data point is at $t = 1000$, and the initial apparent dip in performance can be ignored. Each graph also shows the mean reward as the solid line. See appendix C for variance and standard deviation formulae.

this thesis, and so variances will generally not be shown on future occasions — just the means. One point worth noting is that it is highly unlikely that any particular run generated all the highest points of 5.18(a) or indeed the lowest. In other words, the 36 runs plotted as a single line (in the same way that the mean is plotted), would not yield a set of parallel graphs. Instead, the stochastic nature of the environment (the agent is not at the same point in the physical environment at the same time in different trials) means that each run will tend to produce overlapping reward curves with some unusually high reward values and some unusually low ones. This is despite the averaging of each data point over 1000 time-steps.

Figure 5.19 shows the Input map in the six dimensions of the input space, and the Motor map in the two dimensions of the motor space. The Motor units are shaded according to their position in motor space (see figure 5.20): the sharper the left turn the unit represents, the heavier the shading, and the sharper the right turn, the lighter the shading. As before, each unit in the Input map is coloured the same as the Motor unit for which it has the highest Q-value. The first three plots look similar to before with right turns being prescribed in situations corresponding to left obstacles and vice-versa, but the fourth plot is new and shows the actions that have been discovered by the Motor map. Out of the twenty Motor units, all but three lie either in the top right or bottom left corners corresponding to the fixed actions of the previous experiment. Although each Input unit only requires that there is at least one Motor unit that serves its purpose, in the absence of any other influence, the remaining units of the Motor map will eventually be pulled towards these regions of the action space, providing plasticity in the system is maintained for long enough. In this example there are two stray Motor units that are not used by any Input units and that have not yet been pulled into line by their neighbours. The longer the run continues, the neater the Motor map becomes.

The diagram of the Input map in physical space is not shown because it is similar to that of figure 5.12.



(a) (b) (c) (d)

Figure 5.19: The Input and Motor maps after learning. (a)-(c) represent the Input map and are similar to figure 5.13. Figure (d) is the new plot of particular interest showing the motor map. Motor units are shaded according to their position in the Motor space (see figure 5.20) and Input units are coded according to the Motor unit for which they have the highest Q-value.

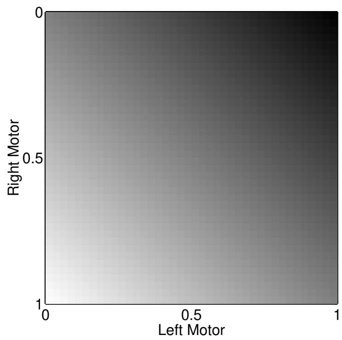


Figure 5.20: Each Motor unit is shaded according to where it lies in the motor space; specifically, how far it lies from the equi-motor diagonal $\langle 0,0\rangle - \langle 1,1\rangle$. The top right corner corresponds to a right turn on the spot, and the bottom left corner to a left turn on the spot. This coding scheme will also be useful in the following experiment where the degree of turn will be more significant.

5.6.3 Changing the reward signal

The optimal behaviour in the previous example was easy to guess at since the faster the agent turns, the quicker the rewarded default behaviour of moving forward can be resumed. To make the optimal actions less predictable, the reward signal is now altered to include a cost for making sharp turns. The reward for the obstacle avoidance behaviour at time t is now given by:

$$r_{obstacle}(t) = r_{basic} + r_{backwards} + r_{angle} \tag{5.8}$$

where:

$$r_{basic} = \begin{cases} 1 & \text{If no sensor activity is above 0.2} \\ -2 \times S1 - 1 \times S3 - 0.5 \times S5 & \\ -2 \times S2 - 1 \times S4 - 0.5 \times S6 & \text{Otherwise} \end{cases}$$

$$r_{backwards} = \begin{cases} -2 \times (M_L + M_R - 1) & \text{If } M_L + M_R < 1 \\ 0 & \text{Otherwise} \end{cases}$$

$$r_{angle} = -\phi(\dot{\theta})$$

where $\dot{\theta}$ is the change in angle of the robot in the last time-step, and the function ϕ is shown in figure 5.21. Note that r_{basic} is just the original reward function of equation 5.1, and $r_{backwards}$ is the component added in equation 5.7 to avoid repetitive behaviour. The function ϕ increasingly punishes sharper turns, the sharpest angle change possible in one time-step being ≈ 0.2 radians. Balancing the three elements of the reward signal is important in order to create the right problem for the learning system to solve. The problem now facing the agent is how fast to turn away from obstacles. A fast turn has the benefit of escaping obstacles more quickly, but incurs the cost of r_{angle}. The system must effectively find the point on the function ϕ that best balances this trade off. The parameters used for this experiment are the same as before.

Figure 5.23 shows a typical solution found by the system. This time the Motor units have polarised into two weaker turns in order to avoid the sting of r_{angle}. The graph of figure 5.22 shows reward over time for this experiment, with the performance of some handcoded solutions involving turns of varying degrees of sharpness also shown by the dotted lines. The original handcoded solution (recall equation (5.5)) involving

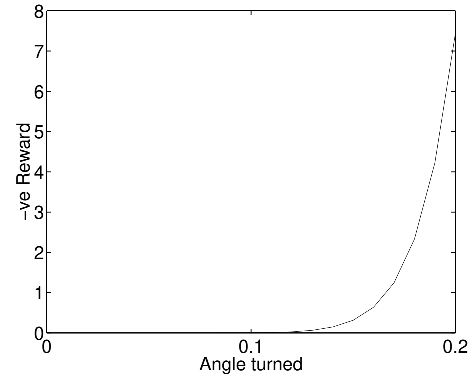


Figure 5.21: $\phi(\dot{\theta})$ plotted against $\dot{\theta}$. $\phi(\dot{\theta}) = (\dot{\theta} \times 6)^{11}$.

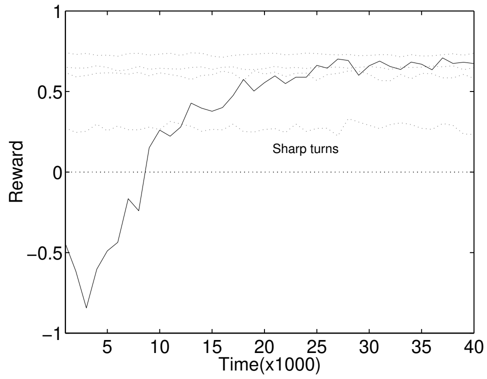


Figure 5.22: Reward over time for the augmented reward signal of equation 5.8. Dotted lines show performance for a number of handcoded (not learned) behaviours involving turns of varying degrees of sharpness. From the top down, these are $\langle 0.25, 1\rangle, \langle 0.5, 1\rangle, \langle 0.4, 1\rangle, \langle 0, 1\rangle$.

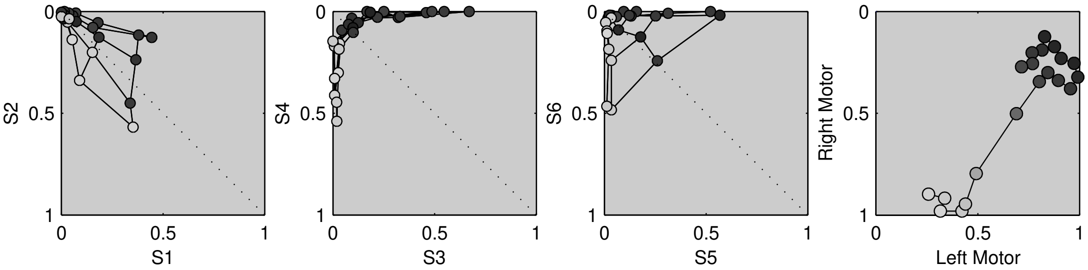


Figure 5.23: The Input and Motor maps after learning. Weaker turns are represented by the Motor map, reflecting the new punishment for sharp turns. The same colour coding scheme as before is used.

the sharpest turns possible is actually the worst performer here, but it can be seen that the learning system itself performed well with respect to the best handcoded strategy found. Figure 5.22 is averaged over only a few runs to indicate typical variability. Note in particular how successive averages (still over 1000 time-steps) do not always improve on the previous reward.

5.6.4 Learning predefined actions

In the next experiment the agent must learn to perform a predefined action for each situation it finds itself in. The system is punished for its distance (in action space) from this 'optimal action' (similar to the experiment of Gullapalli (1990), described in section 4.9), creating a simple RL problem which is half way towards supervised learning. Note that this does not actually constitute supervised learning because the desired output is not explicitly provided. For example, knowing the distance of the proposed action from the desired action does not reveal the direction (in action space), and therefore the location, of the desired action itself. This experiment is to test how well the Motor map can discover a continuous range of values. The setup is similar to before, but now the reward horizon, h, is reduced to 1 since the supervisor provides immediate feedback. This time the task is to learn actions for *all* situations, not just those pertaining to obstacle avoidance. The reward signal at each time-step is simply the negative distance in the action space of the actual action taken from the prescribed action, where the prescribed action is defined as:

$$Prescribed = \begin{cases} \langle 1,1 \rangle & \text{If no sensor activity is above 0.2} \\ \langle 1, 1-max(S1,S3,S5) \rangle & \text{If } S1+S3+S5 > S2+S4+S6 \\ \langle 1-max(S2,S4,S6),1 \rangle & \text{Otherwise} \end{cases} \tag{5.9}$$

The idea is first to discover which side of the robot has higher sensory activation, and then to turn away from that side with a turn that is proportional in steepness to the activation of the highest sensor on that side. Hence a range of actions are required from $\langle 1,1 \rangle$ when there is no sensor activity, to $\langle 1,0 \rangle$ when one of the left sensors is maximal, and $\langle 0,1 \rangle$ when one of the right sensors is maximal.

Since the agent will spend most of its time in open space, it is necessary to increase the size of the Input and Motor maps so that a proportional representation of sensory and motor categories will still leave some units available for the less frequently encountered obstacle avoidance situations. This does not need to have a serious impact on the

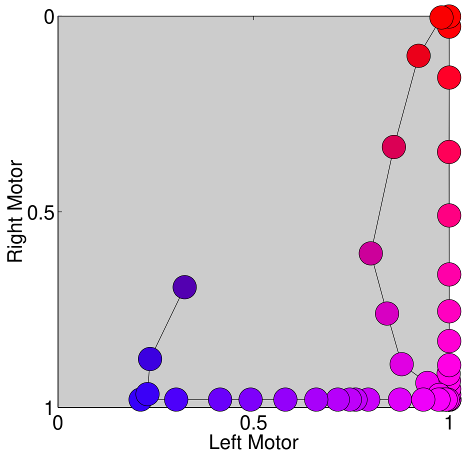


Figure 5.24: The Motor map drawn in the action space with each unit coloured according to its position in this space (see figure 5.25).

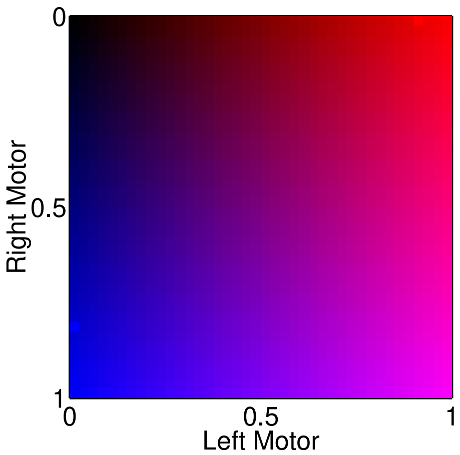


Figure 5.25: Each Motor unit is colour coded according to where it lies in the motor space for use in figures 5.27, 5.28 and 5.29. The horizontal position dictates the amount of red at that point, and the vertical position the amount of blue. This approach has the advantage of giving every point in the two-dimensional space a unique label.

learning speed because the neighbourhoods can also be extended so that where there is redundancy in the representation, units will just learn from each other. For this experiment an Input map of size 7×7 units and a Motor map of 50×1 units are used, with the initial and minimum neighbourhood sizes of the Motor map set to 25 and 1 respectively, with annealing taking place as usual (this time, $f(t) = 0.9997^t$). Because of the simplicity of the reward signal, a final difference is that the more familiar reward calculation of the *first* graph of figure 5.6 is used. Absolute reward values, and not changes in reward are more relevant here.

Figure 5.24 shows the Motor map in the motor space after learning this new task. Now the right and bottom edges of the space are more smoothly represented by the map reflecting the relevance of these regions to achieving the prescribed actions and thus maximising the reward signal. The reward against time averaged over a number of runs is shown in figure 5.26 along with the annealing function (plotted below the axis for convenience) and the line achieved by hardwiring the prescribed action. This last line is not quite zero because of the noise automatically added to the motors.

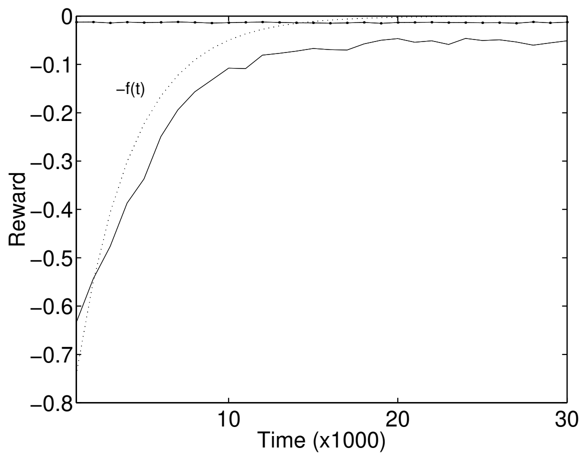


Figure 5.26: Reward over time for learning prescribed actions. The dotted line shows the annealing function used (below the axis for convenience), and the top horizontal line shows the reward of actually taking the prescribed action. This line shows residual error which is the result of the noise automatically added to the motors by the robot simulator.

The results appear satisfactory, but figure 5.27 suggests that so many of the Input units respond to situations corresponding to no sensor activity (and therefore requiring forward movement), that only a handful of units are left over to represent the graded degrees of turn that are also required as part of this task. This is confirmed in figure 5.28 which shows the Motor map as a string of colour-coded units, and which highlights that the bias towards situations involving no sensor activity is present in the Motor map too. For comparison, consider figure 5.29 where the same problem is tackled, except with the default behaviour of moving forward restored. i.e. so the learning system only has to deal with obstacle avoidance situations, with situations involving no sensor activity being taken care of automatically. Now all the Input and Output units can concentrate on the graded turns required for avoiding collisions, and the problem can be represented at a higher resolution with smoother transitions between the behaviours of units. The Output map is not shown here because it is similar to figure 5.24 only containing fewer 'forward' units, and the colour of the Input units can be used to infer the position of their prescribed Output unit by referring to figure 5.25.

In addition to the previously discussed implications of behaviour based control for robustness and reliability, Dorigo (1995) also observes positive implications for the *speed* of learning when a task is broken down in this way. The problem of losing much of

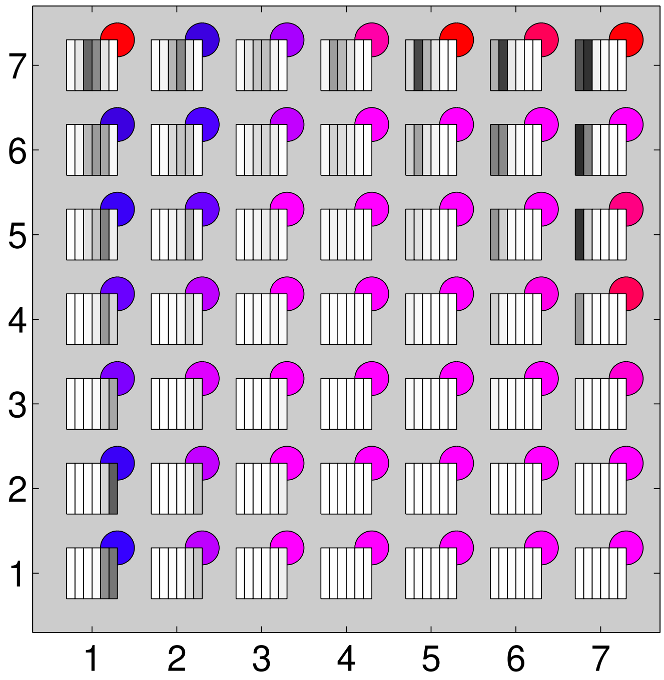


Figure 5.27: Plot of the Input map in topological or physical space after learning the prescribed actions. Each Input unit is colour-coded according to the Motor unit to which it is most strongly connected (see figure 5.24).

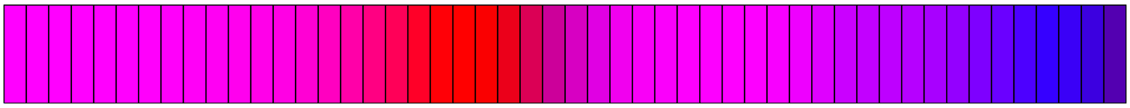


Figure 5.28: Plot of the Motor map in physical space after learning the prescribed actions. Most units provide for the situations that require forward movement.

the Input and Motor maps to the more frequently occurring parts of this particular task emphasises further the utility of breaking the control up into specialised behaviours, this time in terms of representational efficiency.

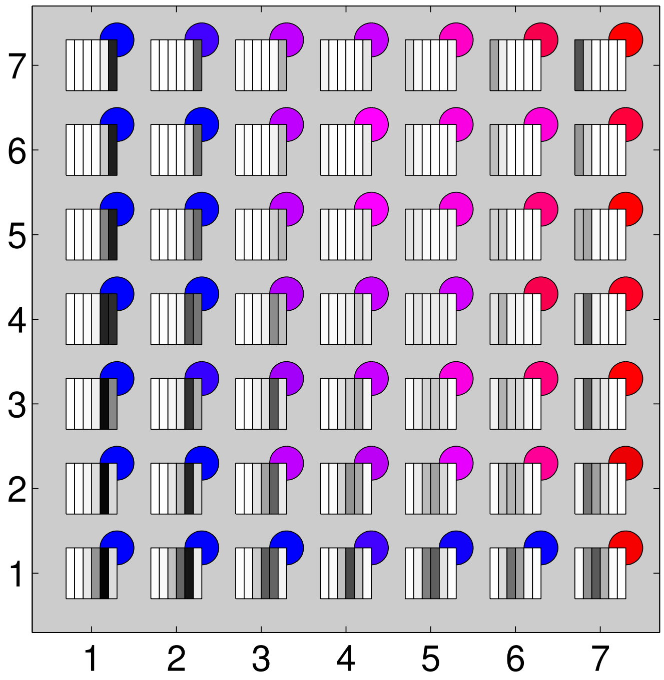


Figure 5.29: Plot of the Input map in physical space after learning the prescribed actions with the default behaviour for moving forward restored. Now the problem of taking graded turns can be represented at a finer resolution.

5.6.5 Multiple behaviours

We now briefly consider the potential applicability of the proposed model to a behaviour based framework. The advantages of breaking global behaviour up into a number of small, cooperating, reactive behaviours have already been discussed in section 3.3. This approach has been used successfully in many applications including Brooks (1986); Mataric (1994); Shackleton and Gini (1997); Maes and Brooks (1990); Mahadevan and Connell (1991) and Dorigo and Colombetti (1994). Following the approach of Mahadevan and Connell (1991), a simple behaviour based experiment is now set up which introduces a new, *goal attraction* behaviour. A single goal can be situated at any point in the environment and the agent is given some goal-sensors in addition to its existing sensors. These sensors do not appear on the real robot and were added to the simulator for the purposes of this experiment. The goal sensors work in a similar way to the distance sensors, with their activation set negatively proportional to the distance between the sensor and the goal, multiplied by the cosine of the angle between them. The range of these sensors is set larger than that of the distance sensors and this allows the agent to perceive a goal from anywhere in the environment. The sensor activity is normalised in the usual way so that a value of 1 corresponds to a goal touching the sensor, and a value of 0 corresponds to a goal either in the opposite corner of the environment to the agent (i.e. maximum distance away), or a closer goal subtending a large angle with that sensor[11]. There are eight goal sensors in all, in identical positions and oriented at identical angles to the six distance sensors of figure 5.3, with the addition of two rear sensors (see Michel (1996)).

A new reinforcement signal, r_{goal}, is calculated at each time step as the sum of the front six goal sensors minus the two rear goal sensors. This value is then multiplied by three so that it balances the other reward values. If a goal is reached, r_{goal} is set to 2 for the next h time-steps, and the goal is immediately moved to a new part of the environment. This part of the reward signal is to reinforce collecting the goal, which would otherwise not be favoured since it indirectly results in the goal being moved to a distant location. The r_{goal} signal also makes use of the usual punishment for moving backwards (equation 5.7) and an additional component is added that rewards movement in a general forward direction while seeking goals: $(M_L + M_R - 1)$, if $M_L + M_R > 1$.

This is to avoid repetitively turning on the spot to align with a goal without making any forwards progress towards the goal. These exact details are not vitally important, the emphasis being that the reward signal is composed and balanced in such a way that the desired global behaviour can emerge. This may not always be a trivial task, but neither does it appear to be one with just a single set of magic values. A suitable reward signal must simply be discovered through trial and error on the part of the designer. The important points here are that the agent is rewarded for increasing its goal sensor activity, and punished for moving backwards or making on-the-spot turns. It would be nice to be able to abstract away from this level of detail, and simply reinforce the agent when a goal is reached, but this seems unlikely to happen often enough by chance to bootstrap the learning process quickly enough in a practical application. The long delayed rewards would also preclude the kind of simplified learning mechanism used here[12].

[11] Given the different circumstances that can generate the same individual sensor reading, it is necessary to be aware of potential problems with perceptual aliasing.

[12] Using a local reward signal is advocated by Kaelbling et al. (1996), and implemented in the form of *progress estimators* in Mataric (1994, 1997). See the discussion on delayed rewards in section 2.9.2.

Now the initial experiment of section 5.6.1 is augmented in the following way: The goal sensors are added to the robot and at each time-step, control is given to one of three behaviours according to the following criteria:

If *any* distance sensor > 0.2 then Behaviour = Obstacle Avoidance

else if *any* goal sensor > 0.2 then Behaviour = Goal Attraction

else Behaviour = Default Forward

At each time-step both $r_{obstacle}$ and r_{goal} are calculated and fed to their respective behaviours. A new pair of Input and Motor maps are created to represent the goal attraction behaviour, with the new Input map receiving input from the eight goal sensors. At each time-step, the behaviour currently in control identifies the input stimulus and takes an action in the usual way. Now two behaviours (plus the default behaviour) share control of the agent (with obstacle avoidance taking priority over goal attraction) and both must learn an appropriate mapping from their own particular inputs to their own motor outputs in order to maximise their own reinforcement signal. The hope is, that if each behaviour learns its task correctly, then the interaction of the behaviours will result in the agent being able to wander around its environment, finding goals, but avoiding obstacles en route. The architecture is illustrated in figure 5.30 and the test is now whether or not this extended system can cope with learning multiple behaviours simultaneously.

As learning progresses, the agent develops the kind of behaviour shown in figures 5.31(a) and 5.31(b). As the agent approaches an obstacle, it has clearly learned to take the usual avoidance action. However, as soon as the agent has turned sufficiently for the obstacle to be out of sight, the goal attraction behaviour is invoked, which has learned to turn and move towards the goal. These behaviours alternate in a cooperative manner to guide the agent around obstacles towards its goal. This is why the r_{goal} signal included an incentive for making forward progress. If it had not, then the interaction of the two behaviours around obstacles would have sometimes resulted in repetitive

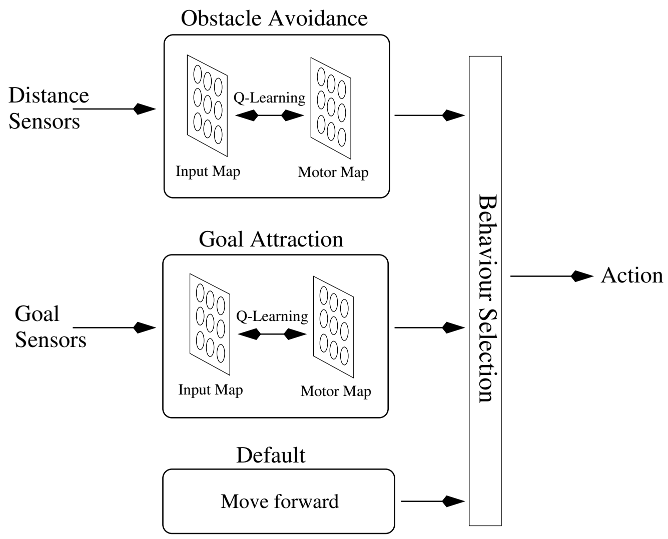
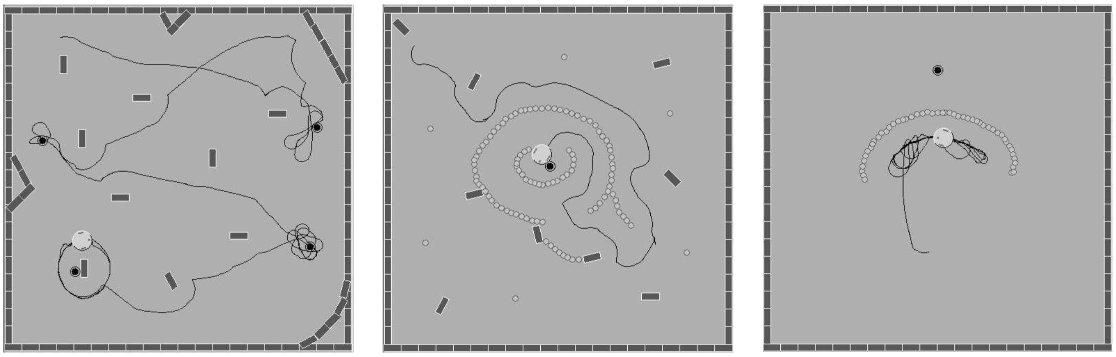


Figure 5.30: Learning actions in a continuous action space within a framework of multiple behaviours.

turning cycles first towards the goal and then away from the obstacle, with no overall progress.

Of course most of the work is done by the two hand-crafted reinforcement signals, and the superficial — even if generally effective — nature of the agent's intelligence is highlighted in figure 5.31(c) where the agent is unable to take a long term view of its aspirations. The learning time of these trials is typically slightly longer than the standard learning time of figure 5.17, but of the same order. The maps themselves look similar to the maps seen so far, and are not duplicated here.

One point worth mentioning is that each behaviour must have sufficient exposure to relevant situations during the critical learning phase — i.e. when the exploration is still high. For example, with few obstacles around, the agent could spend the vast majority of its time involved in the goal attraction behaviour, and never have enough time in the collision avoidance behaviour to become competent. To address this, the goal attraction behaviour was toggled on and off. While this attention flag was off, the agent behaved as if it did not have the goal attraction behaviour in its repertoire, allowing the agent to gain sufficient exposure to its other behaviour.



(a) (b) (c)

Figure 5.31: Learning actions in a continuous action space within a framework of multiple behaviours. The agent's path is drawn with a line. Goals are represented as filled circles. Multiple goals indicate subsequent positions of the same goal. Obstacles are represented by the brick shaped objects and the light coloured circles.

Behaviour Composition

Other combinations of behaviours are fun to play with, and some of these are illustrated in figure 5.32. Apart from the eight distance sensors, the Khepera robot also has eight identically positioned and oriented light sensors. These sensors have a greater range than the distance sensors and their activations are calculated in a similar way to the goal sensors, although the light sensors are actually supported on the real robot. Figure (a) shows the effect of replacing the goal attraction behaviour with a light attraction behaviour in an environment consisting of four light sources, each surrounded by a ring of small obstacles. The reward signal for light attraction is calculated in a similar way to r_{goal} except that only the front six sensors are used, and the contribution of each sensor to the signal is weighted according to the position of the sensor. Central sensors are weighted more heavily. The robot learns to head towards light sources until the obstacles surrounding the source are detected, at which point the obstacle avoidance behaviour turns the robot away from the obstacles and the light, and then the default forward behaviour takes the robot back off into the environment.

Figure (b) shows the effect of modifying the obstacle avoidance behaviour so that higher activity is tolerated on the peripheral sensors before this behaviour is invoked.

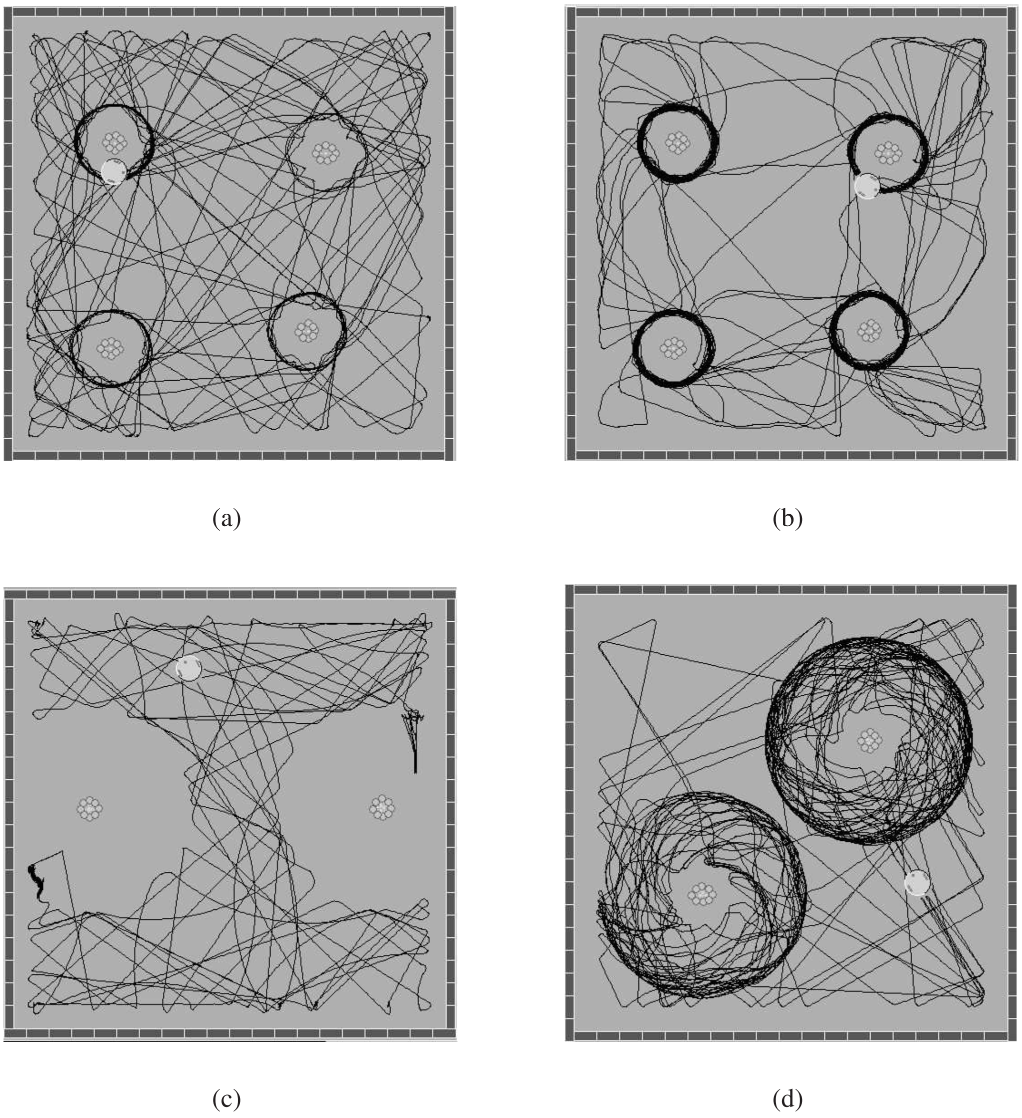


Figure 5.32: Compositions of learned behaviours involving light sources.

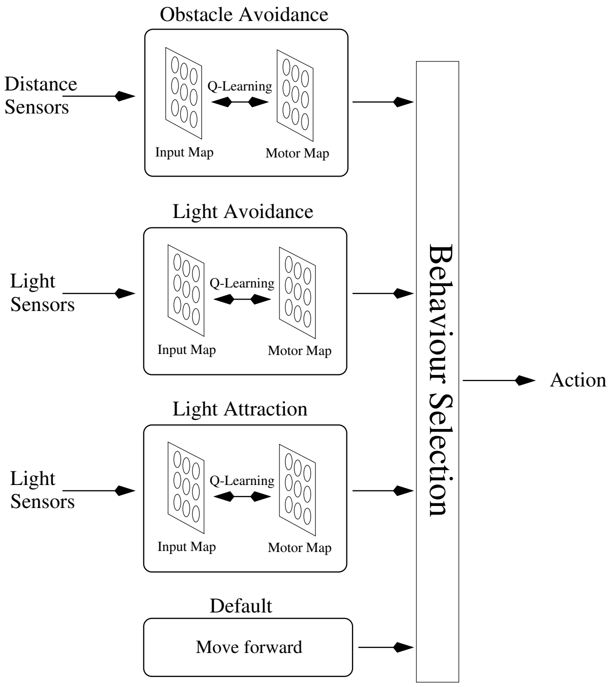


Figure 5.33: Four behaviours contributing to the global behaviour of figure 5.32(d) prioritised from top to bottom.

Now the balance between the light attraction and collision avoidance behaviours is changed so that the robot alternates between the two in the presence of both obstacles and a light source. The effect is that the robot tends to circle the light sources as the two behaviours jostle for control. Figure (c) shows the effect of substituting light attraction with light avoidance. The predictable result is that the robot turns away from the source when it gets close enough to trigger the behaviour. Finally, figure (d) demonstrates how attraction and aversion forces may be combined. Now both light attraction and light aversion behaviours cooperate in encouraging the robot to maintain a constant distance from the light sources. In this example, there are four prioritised behaviours, as illustrated in figure 5.33.

Sharing the Motor map

Different behaviours may share a common Motor map, as illustrated in figure 5.34. First the current behaviour is identified using the same prioritising rule as before, and

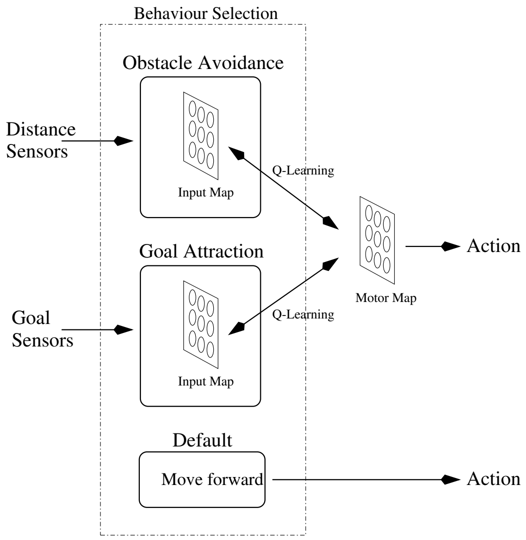


Figure 5.34: Behaviours sharing the same Motor map.

then the relevant Input map selects a Motor unit from the common Motor map. The Q-values are effectively stored on the connections between the Input maps and the Motor map, so there is no conflict here. However, the units of the Motor map must now service a wider range of situations belonging to different behaviours, and this may entail an extra burden if there are many conflicting actions to be learned. But conversely, sharing a Motor map should facilitate the sharing and re-use of actions across behaviours. Since learning actions is potentially the most expensive problem the system has, the advantage of re-use may outweigh the potential conflicts associated with sharing. This particular extension to the proposed model[13] has not been investigated, and is suggested as an avenue for future work.

5.6.6 Plasticity

Since the intention is that unsupervised learning will allow compensation for sensor drift, changes in the environment, circumvention of minor internal or physical failure etc, it is worth briefly considering the effect of damage or interference to the system

[13]The term *proposed model* is used throughout this thesis to refer to the algorithm and architecture introduced in this chapter.

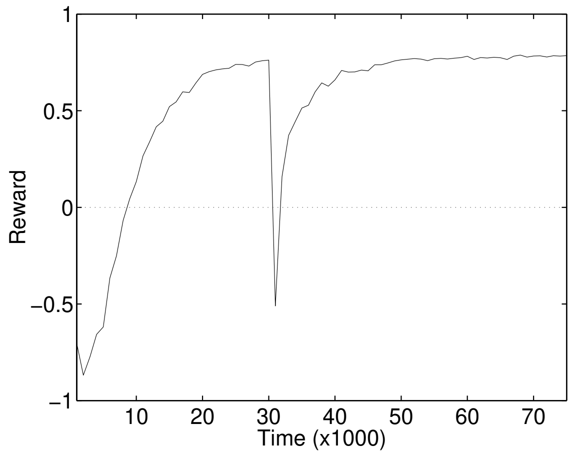


Figure 5.35: Reward over time for the experiment of section 5.6.1, with left-right stimuli swapping at $t = 30000$.

after or during learning. The next test involves running the basic experiment of section 5.6.1, except that after learning is complete, the sensors are swapped so that the left sensors are fed to the right inputs (of the Input map) and vice-versa. A small amount of exploration and low learning rates and neighbourhoods are maintained so that the ability to adapt is preserved throughout the experiment. The graph of figure 5.35 shows the result with Q-values being re-established quickly compared with the initial time it took to acquire competence. Since both the Input and Motor units are already in place, the Q-values are the only part of the system that need to change.

If the Input map is damaged, for example by removing some of the units, the map can easily adapt, distributing the remaining units appropriately. However, if the representation of the *action* space is compromised by removing portions of the Motor map, then recovery tends to be more problematic. Small, residual exploration and learning rates may be ineffective in rediscovering useful actions that lie over canyons on the reward surface, and for the system to compensate for severe damage to the Motor map the learning process may need to be completely re-stimulated. Learning actions is the hardest part of this system and, depending on the task, may require large amounts of exploratory noise.

5.6.7 Regression problems

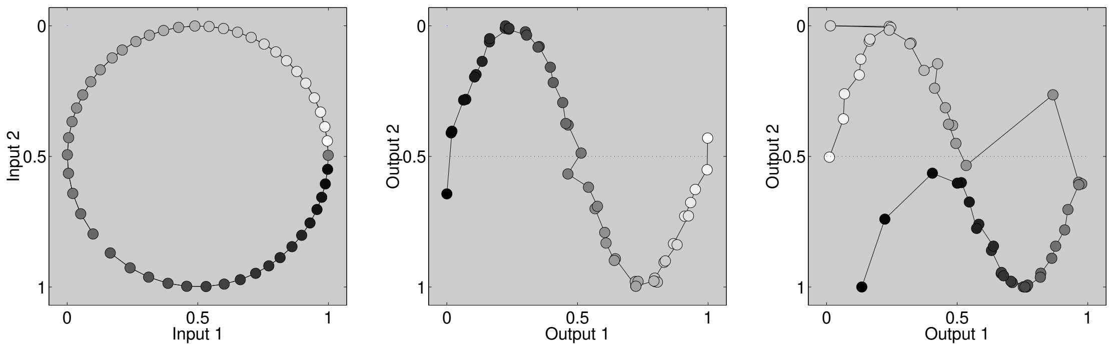
This section moves away from the robot simulator environment and briefly considers the architecture in a more abstract context. In its most general form the system learns a mapping between two spaces of arbitrary dimensions using an external scalar signal for guidance. Section 5.6.4 on learning predefined actions introduced the idea of an immediate act-reward-learn cycle, with the reward proportional to the distance in action space from a prescribed action. Although the concept of an optimal action exists here, the system itself only receives a scalar error signal. Following this approach, and to test the flexibility of the Output map, the system is now set a number of regression style learning tasks. Each task will focus on a different type of mapping.

The first task is to learn a mapping from a two dimensional Input space to a two dimensional output space. Both spaces are the unit square. Inputs, $\langle \frac{1+cos(\theta)}{2}, \frac{1+sin(\theta)}{2} \rangle$ are generated with θ selected randomly and uniformly from the interval $[0, 2\pi]$. Each input pair is associated with a prescribed output, $\langle \frac{\theta}{2\pi}, -0.5 \times (sin(\theta) - 1) \rangle$. These functions effectively map a circle to the sin function within the normalised spaces. The system is presented an input stimulus and must learn to recognise that stimulus and produce the correct output on future occasions. Reward is given immediately ($h = 1$), and is equal to the negative distance in output space between the prescribed output and the actual output generated by the system. Both the Input and Output maps are represented by a 1-dimensional Kohonen map, each comprising 50 units. Annealing takes place in the usual way, with large initial neighbourhoods, and high initial learning rates (see table 5.36).

Figure 5.37(a) shows a typical Input map plotted in weight space after learning, and figure (b) shows the corresponding Output map in the output weight space. The Output map is shaded according to unit index (in terms of its topological position in the map), and each unit of the Input map is coloured according to the Output unit for which it has the highest estimated expected reward. Figure (c) shows an Output map from a second trial in which the space is not mapped contiguously. There are also deviant units in this map, but only units on the curve will be used by the Input units. Note also that the Output maps are more irregular than the Input map. This is a direct consequence of the random exploration that is required to drive the discovery of actions. The deviant

Parameter	Value
Input map size	50×1 units
Motor map size	50×1 units
Input map neighbourhood size, *IN*	$20 \times f(t)$
Motor map neighbourhood size, *ON*	$20 \times f(t)$
Q-learning rate, α	$f(t)$
Learning rate of Input map, *IL*	$0.3 \times f(t)$
Learning rate of Motor map, *OL*	$f(t)$
Probability of Q-learning Exploration, *p*	$f(t)$
Probability of Large Motor Exploration, *q*	$f(t)$
Max. Exploration distance around Motor unit, *MA*	$f(t)$
Annealing schedule, $f(t)$	0.9998^t

Figure 5.36: Table of parameters for regression problems.



(a) Input map

(b) Output map

(c) A second Output map produced by a different trial. The Input map is not shown (similar to (a)).

Figure 5.37: Reinforcement learning of a mapping from input to output space.

units in the second Output map are a legacy of the initial stages of this exploration. In contrast to the Output map, the Input map is formed in a reactive rather than interactive manner, in the sense that there is no explicit exploration of the space.

The average error — interpreted as the negative of reward — was reduced to around 0.03 for this experiment, but there is the usual trade-off to be made between final

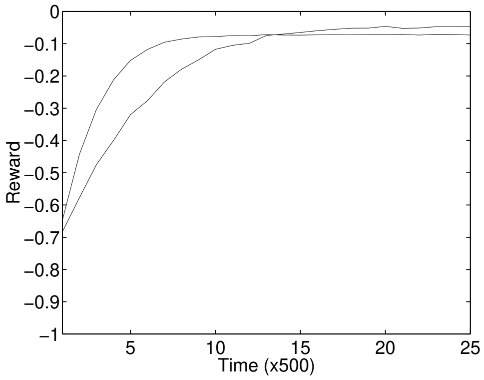


Figure 5.38: Reward over time for different annealing rates. The steepest graph uses an annealing schedule of $f(t) = 0.9995^t$, and the other graph a schedule of $f(t) = 0.9998^t$. The actual schedules are not overlaid because they are of a very similar shape to the reward graphs themselves.

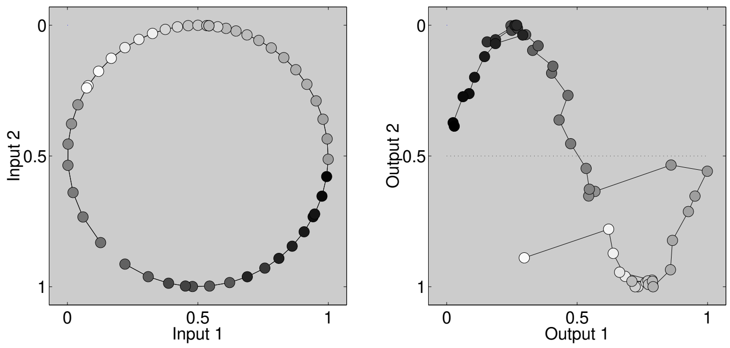
performance and learning speed. Indeed learning may be expedited considerably for a modest sacrifice in final performance, as illustrated in figure 5.38 which shows two error curves for different annealing rates. It is possible that a more judicious annealing schedule (of a form different to $f(t) = c^t$) could attain the best of both runs.

Figure 5.39 shows maps for the same problem except that instead of θ being selected uniformly from the range $[0, 2\pi]$, it is selected according to the probability distribution $\frac{1}{4}|sin(X)|$ over the same range. Both the Input and Output maps reflect this distribution with higher densities of units at points corresponding to $\theta = \frac{\pi}{2}$ and $\theta = \frac{3\pi}{2}$.

The mapping between the Input and Output maps need not be one-to-one. Figure 5.40 shows the same problem (returning to the equal distribution), except that the input pair $\langle \frac{1+cos(\theta)}{2}, \frac{1+sin(\theta)}{2} \rangle$ is mapped to the output pair, $\langle \frac{\theta'}{2\pi}, -0.5 \times sin(\theta') - 1 \rangle$, where

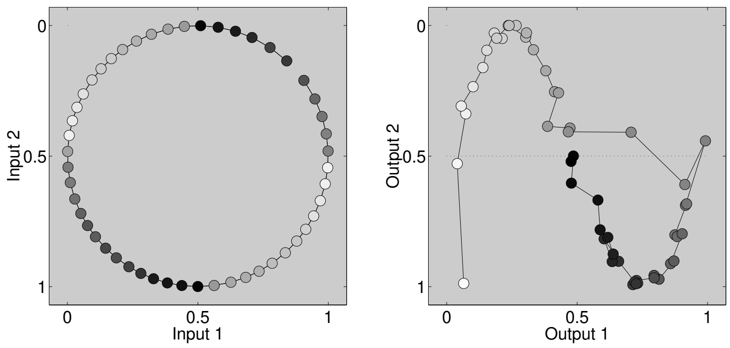
$$\theta' = \begin{cases} 2\theta & \text{If } 2\theta < 2\pi \\ 2\theta - 2\pi & \text{Otherwise} \end{cases}$$

The effect is that one revolution of the circle in the input space is now mapped to two periods of the sin curve, so that Output units are re-used by different areas of the Input map.



(a) Input map (b) Output map

Figure 5.39: The density of units in both maps reflects the distribution of input stimuli.



(a) Input map (b) Output map

Figure 5.40: Input map shows re-use of Output units.

Neither must the mapping between the Input and Output maps be contiguous[14], although the more contiguous the mapping the more efficiently the problem can be learned since the more appropriate it will be for units to learn from their neighbours. Figure 5.42 shows the Input and Output map for a problem in which inputs are taken from the distribution $\langle x, 0.5\rangle$ with x randomly and evenly drawn from the range $[0,1]$. The constant of 0.5 is used here purely as an implementational convenience. The cor-

[14]By *contiguous* it is meant that neighbouring Input units prescribe neighbouring Action units. The word *continuous* has been avoided because continuity implies that the function is smooth, which it clearly need not be.

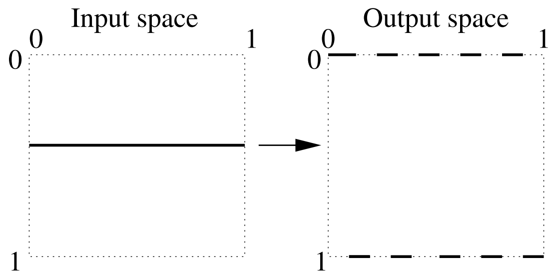
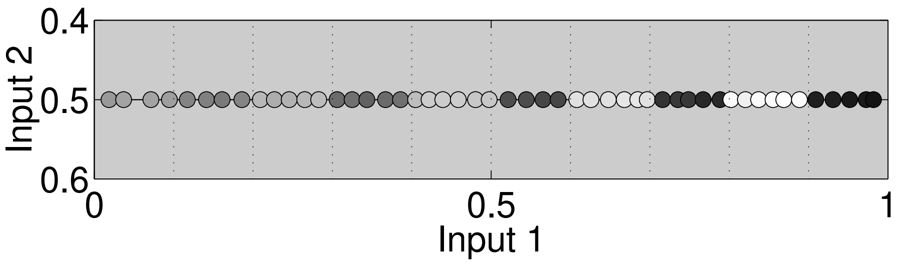


Figure 5.41: Function to be learned. Each point on the line in the input space must be mapped to the point on the line in the output space with a corresponding 'x' value.

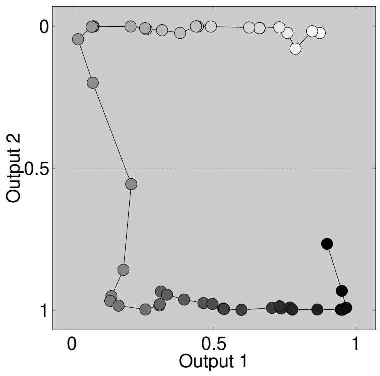
rect output is defined as $\langle x, 0\rangle$ if $0 \geq x < 0.1$ or $0.2 \geq x < 0.3$ or $0.4 \geq x < 0.5\ldots$ and $\langle x, 1\rangle$ otherwise (see figure 5.41). In figure 5.42 both spaces are mapped contiguously with neighbouring units representing nearby regions in the space, but the mapping between the two spaces is non-contiguous; i.e. at some points, neighbouring Input units prescribe very different positions in the Output map. Figure (c) shows the connection strengths or Q-values between each Input unit and each Output unit. Black corresponds to the highest estimated expected reward and white to the lowest. The pattern is consistent with the two maps in (a) and (b) and illustrates neighbourhood similarities. In particular it is evident that most neighbouring Input units (represented by adjacent vertical lines on the plot) maintain very similar Q-values for every Output unit. Similarly the Q-values associated with any two neighbouring *Output* units (represented by adjacent horizontal lines on the plot) also look similar. These features are exploited later in appendix F in order to abstract a compressed representation of the Q-table.

5.7 Summary

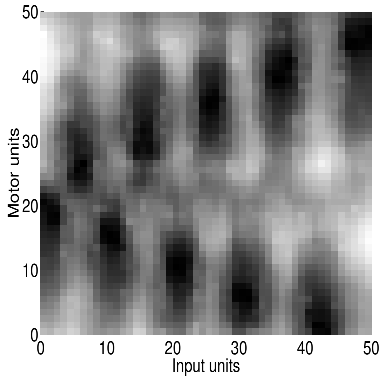
This chapter has outlined the proposed model of this thesis within the context of a number of different experiments. In particular we have attempted to demonstrate the suitability of the model for interactive, delayed reward problems such as those typically encountered in robot learning, as well as the potential applicability to the behaviour based domain. We briefly considered robustness to internal damage and, implicitly, robustness to non-stationary environments, although this latter topic will be given a more thorough treatment in chapter 8. We have also shown that the model exhibits



(a) Magnified plot of the Input map in weight space. As usual, Input units are coloured according to the Output unit with which they maintain the highest Q-value.



(b) The solution discovered by the Output map. Each unit is coloured according to its physical *index* within the map.



(c) Connection strengths between every input-output pair. The higher the Q-value, the darker the shading.

Figure 5.42: Solving the non-contiguous mapping problem.

flexibility in that it can achieve a range of different mappings of both a contiguous and non-contiguous nature.

The topology preservation offered by the SOM has also been shown to be useful for expediting learning by facilitating the efficient distribution of reward information around the Q-table. Furthermore, providing the neighbourhood function is reduced during the course of a trial, we have seen that topological Q-learning need not interfere with the learning of even significantly non-contiguous mappings.

This chapter has presented an introduction to the model which is analysed in more detail in the following chapter. Subsequent chapters will also compare the features of the model with those of existing techniques, with a view to determining how well the desirable criteria referred to at the beginning of this chapter have been satisfied. We will also consider the limitations and scalability of the model, as well as provide a more detailed investigation of parameters.

CHAPTER 6

Analysis

6.1 Introduction

The experiments of the previous chapter raise a number of issues that are now discussed. In particular, we look at the achievements of the proposed algorithm in terms of the desirable model properties summarised at the beginning of chapter 5, and how the model compares with some of the alternative approaches to representation and generalisation discussed in chapters 2 and 4.

For analysis purposes it is convenient to consider the proposed architecture as consisting of three main functional components: The Kohonen mapping of the input space, the mapping of the output space, and the reinforcement learning between the units of the two maps. The discussion aims to address issues of performance, convergence, stability, plasticity, complexity, limitations, and applicability, as well as the assumptions under which these analyses are relevant. Splitting the model into these three components will turn out to be useful since existing results pertaining to Kohonen's Self-Organising Map and to RL may be appealed to directly. The implications of the interactions between these components are also considered.

6.2 The Input Map

The discussion begins with a review of the features of the Self-Organising Map (SOM) and their implications for the formation of the Input map within the context of the proposed model. Firstly, a number of features of the SOM are considered which may explain its popularity:

❶ Simple and easy to implement with a short processing cycle.

❷ Intuitively appealing with easy visualisation.

❸ Most widely used unsupervised learning method, so a large corpus of empirical data exists.[1]

❹ Straight forward dimensionality reduction technique.

❺ Categories are effectively created, altered *and destroyed* during training because of neighbourhood influences. Some clustering techniques — for example the ART family of networks (Carpenter and Grossberg, 1987b) — only allow creation and adaptation, with no balancing destructive force.

❻ Biological parallels make it interesting to study.

There are also a number of significant drawbacks to the algorithm, and these are now considered in some detail with particular emphasis on how they impact the model proposed in chapter 5.

6.2.1 Lack of energy function and convergence proof

The SOM lacks a convergence proof beyond the one-dimensional case and has been proved to lack a continuous energy function (Erwin et al., 1992). These shortcomings conspire against a principled comparison and analysis of network size, input coding,

[1] For a list of thousands of papers based on the Kohonen map, see (Kangas and Kaski, 1998) (also URL at (SOM-database, 2001)), and for a book of varied applications and analyses see (Oja and Kaski, 1999).

annealing schedules, network dimensionality and learning rates. Knowing under what conditions, and how quickly, convergence to stable minima may occur has obvious benefits. In an ideal world it would also be nice to have a measure of topological preservation. The lack of such proof components is an undisputed drawback of this model. However, alternatives to the standard Kohonen learning rule have been suggested that do yield a continuous Energy function. Heskes (1999), for example, proposes selecting the winner by:

$$Winner = argmin(i) \sum_j h_{ij} \|\vec{x} - \vec{w}_j\|^2 \tag{6.1}$$

instead of

$$Winner = argmin(i) \|\vec{x} - \vec{w}_i\|^2 \tag{6.2}$$

where $\vec{x}$ is the input vector, $\vec{w}_i$ is the weight vector of unit i, and h_{ij} is the neighbourhood function at unit i from unit j. Heskes shows that a continuous energy function with a continuous gradient is the result of using equation (6.1) instead of (6.2). It is then possible to make more quantitative statements about convergence, such as transition times between disordered and ordered states (Heskes, 1996).

In some ways, the easy and intuitive visualisation of the SOM combined with the wealth of empirical data available on the algorithm help compensate for the lack of rigorous analysis. Since any algorithm, no matter how well principled and analysed, is going to require *some* intuitive tuning before it can be imported into a larger application, these properties will always be useful. Of course, to have a sound theory *and* an intuitively appealing algorithm leading to a wealth of empirical knowledge would be the best of all possible worlds. With this in mind, the elastic net of Durbin and Willshaw (1987) deserves to be mentioned. Although not receiving the same popularity as the SOM, the algorithm effectively provides a principled version of the SOM.

6.2.2 No principled way of setting network parameters

The lack of a principled method for setting the network parameters is a drawback of the SOM which is aggravated by the lack of a theoretical foundation. However, this particular problem may be no more than an irritation. It is rarely reported, for example, that a suitable set of parameters was difficult or impossible to find, just that it had to be empirically sought with a typically small amount of trial and error on the part of the experimenter. Moreover, as in the case of the experiments of chapter 5, it often appears that a range of parameters will tend to yield good results.

Some of the symptoms of annealing the neighbourhood too fast that were discovered during the development of the experiments of this thesis were *stranded* units (which are rarely active), unfaithful unit distribution and twisted or *disordered* states. The first two circumstances may easily go undetected, and care is required to avoid them. However, in the context of the problem of mapping the input space in a reinforcement learning problem, none of these three symptoms are in danger of pathologically disrupting the system. Stranded units are wasteful, but do not interfere with performance in any other way. The same is true of uneven distributions, although this may also decrease the benefit of learning Q-values from neighbouring units. Twists in the network are generally more easily detected (visually), but even when they occur they only result in local degradations in the quality of the topology. We do not expect this to represent a serious problem, since the topology preservation is used solely to expedite learning reward values, which should still converge if there is *no* topological structure in the state representation at all (as is usually the case in Q-learning for example). These arguments are borne out by empirical observations made during the course of the experiments of chapter 5.

One key parameter of the SOM is the dimensionality of the network. This must be specified a-priori and relies on the designer having a sensible intuition regarding the inherent dimensionality of the data. The worst consequence of choosing too low a dimension map is poor topological preservation, and therefore diminished benefit of the neighbourhood Q-learning (see figure 6.1). Choosing too high dimension a map may lead to a much larger network than is really necessary. However, although a large number of states is generally to be avoided with Q-learning, the neighbourhood

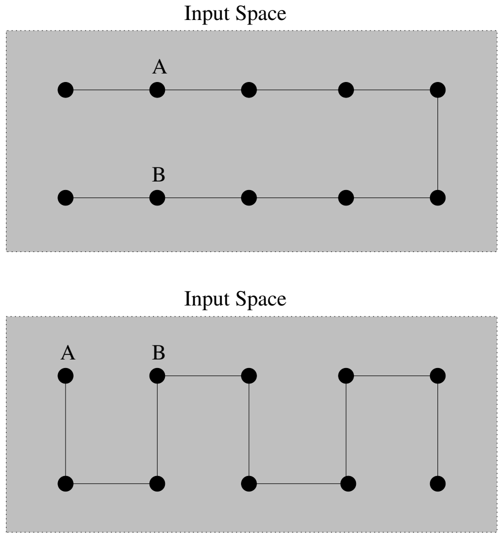


Figure 6.1: Two alternative mappings of a two-dimensional space by a one-dimensional network. In each case the Q-values of A and B cannot learn from each other's updates because they are too distant in the physical topology of the network, despite being close in the input space.

learning employed here will tend to allow large groups of redundant units to learn together.

The dimensionality of a data set can be estimated by a variety of techniques (Barnsley, 1988), although necessarily these techniques assume access to some sample data. However, in the current context such data is unlikely to be available. In a chicken-and-egg fashion, the input data is dependent on a behavioural policy which itself cannot be optimised until some input data is provided. We could guess at an optimal behavioural policy, set the system going, sample some input data, and then calculate an appropriate dimension for the network before starting the run proper, but the dimensionality of the data may change as the policy is adapted on-line. In the case of the robot experiments of chapter 5, the dimensionality of the input appears to go down as reinforcement learning takes place and the set of situations the agent finds itself in becomes smaller and more ordered. In practice one and two dimensional maps tend to be prefered in the literature and these were indeed found to be adequate for the problems considered so far.

6.2.3 Unfaithful distributions

The Kohonen map is sometimes criticised for its unfaithful distribution of units in the input space (Bishop et al., 1998; Li, 1999), with the map tending to under-sample high probability regions, and over-sample low probability regions (Hertz et al., 1994). Again, in the context of the current application, the worst consequence of this is a small reduction in efficiency, as slightly larger maps are required than we might theoretically expect.

Hulle (2000) suggests ways of producing faithfully distributed topographic maps, and goes on to apply the techniques to a number of problems including density estimation. However, in the context of the experiments performed here, it is far from certain that a faithful distribution would actually be the ideal choice. For example, a particularly intuitive idea would be to bias the distribution of units towards regions of input space which vary the most with respect to the reward signal, as it is these regions that require the highest resolution representation (see section 9.2.2). A specific implication of non-equiprobable firing rates is encountered in appendix F.2.6.1.

6.2.4 Assumption of stationarity

As the learning rate and neighbourhood of the SOM are annealed, categories become fixed and changes in the environment can no longer be reflected in the map (in comparison with say the ART network (Carpenter and Grossberg, 1987b) which always allows new categories to be created). Maintaining a small amount of residual plasticity may allow for small adaptations in the input distribution, although coping with a moving target in the Output map may be more challenging. The problems caused by dynamic environments are considered in more detail in section D.5 and in chapter 8.

6.2.5 Curse of dimensionality

The price for the simplicity of the SOM is paid in its generalising power. The representation scheme is local in that each weight only pertains to one part of the input space.

As the size of the space increases, so does the number of free parameters. In contrast, the weights of a feedforward network all apply to *all* parts of the input (and output) space. Chapter 8 investigates the implications of this distinction in detail with specific reference to generalising power, discovering both advantages and disadvantages to both distributed and local representations. As an example, consider the feedforward network used by Tesauro to map backgammon board positions to expected reward. In this instance the input space had two-hundred dimensions with significant and non-trivial interactions between the dimensions. We would clearly expect the SOM to be an inferior choice for generalising over the state space of this problem, in which single input lines can have highly non-linear implications for the reward function — a feature not detectable with the standard Euclidean distance measure of the SOM learning rule.

6.2.6 Parallelisation difficulties

One relatively minor implementational issue is that of parallelisation which may become relevant in real-time applications involving large maps. If our aim is to run different units on different processors then the problem with non-local exchange of information, such as that required for selecting a global winner, is that the interconnectivity required between units becomes large, or else a bottleneck is created[2]. To address this, a hierarchical winner selection such as that illustrated in figure 6.2 could reduce the number of connections required from $O(N^2)$ to $O(Nlog(N))$ where N is the number of computing units (Kohonen, 1995). Although scalability of the Kohonen network is not an immediate issue with respect to the experiments performed so far, it is always useful to bear this issue in mind. See (Kohonen, 1995) for parallel hardware implementations of the Kohonen network and (Rojas, 1996) for discussions on parallelisation of neural algorithms in general.

[2]"The important point in any parallel implementation of neural networks is to restrict communication to local data exchanges" (Rojas, 1996, p 449).

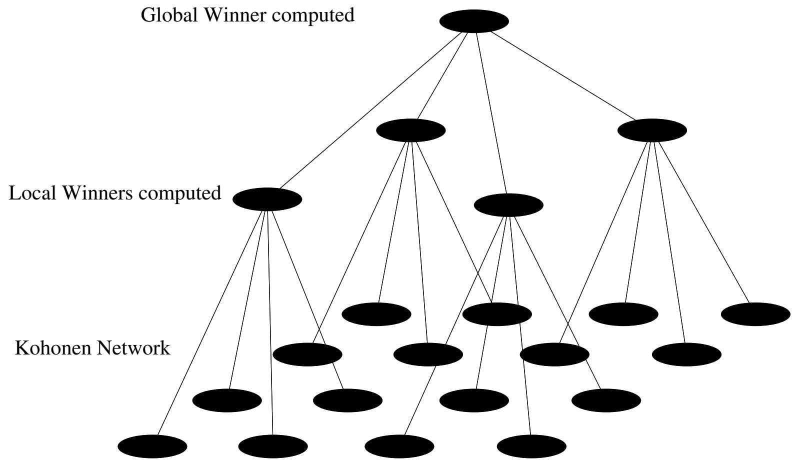


Figure 6.2: First a set of local winning units is identified and then a global winner is selected from this set. Reverse connections propagate control information back to the network units.

6.2.7 Alternative approaches to representing the input space

The question now arises as to how other methods for representing the input space compare with the SOM with respect to some of the advantages and disadvantages discussed above. This section is brief since the primary interest of this thesis is the use of the SOM in the *output* space, which is analysed later in the chapter. See Kaelbling et al. (1996) for a more detailed review of the large body of techniques that exists for state space generalisation.

Hand decomposition of input space

As suggested in section 2.10.1, the simplest approach would be to divide up the state space into handcoded regions and then just treat each region as a single state. The Voronoi tiling of the SOM achieves the same goal, but within a dynamic framework with flexible sized tiles. The representation in the handcoded approach becomes too costly as the number of dimensions of the input data increases. If there are just six real valued sensor readings in the range $[0, 1]$ and the space is populated with tiles of width 0.1, there are still a million states, and in the active regions of the input space the representation would still be at a lower resolution than achieved with a small SOM. Fewer states could be used, for example by just populating the state space with a small number of randomly positioned units. In fact initially this setup was used for the robot learning problems of chapter 5, but performance was poor because of gross over generalisation of the Q-function. Some form of dynamic approach is strongly suggested.

k-means

K-means is analogous to the Kohonen mapping without topology preservation, and thus offers a simpler alternative. K-means has been formulated in both batch (Lloyd, 1982) and on-line (Moody and Darken, 1989) form, so support for a continuous adaptive response to dynamic environments is provided.

Adaptive Resonance Theory

The ART networks (Carpenter and Grossberg, 1987a) were devised to address the stability/plasticity tradeoff in clustering problems. Each unit of the network can be thought of as a category prototype of some fixed receptive field. New vectors are classified either by adding them to a current category if they happen to fall within its receptive field, or by adding a completely new category to classify the rogue data point. In this way, stability is provided because existing prototypes are only updated if the new data is close to those prototypes. But plasticity is always maintained through the ability to add new prototypes.

In contrast, SOM categories that are not actively maintained by activation in that region of the input space are slowly (re)moved through neighbourhood learning in favour of the more salient representations of their neighbourhood. This balance of creative and destructive forces provides a very general framework for mapping a space, and the inability of the ART network to forget categories as well as create them could be considered a weakness in some contexts. For a system with finite resources operating in a world that constantly changes, forgetting ought to be considered an intrinsic part of learning.

ART offers no topology preservation. It also attempts to build clusters of the same size (controlled by the radius of the receptive field) and thus does not immediately lend itself to density modelling. However, the permanent ability to add new categories as required may be an advantage over the SOM in some circumstances, although there is no reason why new units could not be dynamically added to a Kohonen map as well, and judicious adjustment of the annealing schedule can always allow for some plasticity in the network. Li and Svensson (1996) choose an ART network over a Kohonen network to categorise the input space in a navigation problem because of the perceived inability of the latter to operate simultaneous learning and operation phases. However, results from chapter 5 suggest that this conclusion is questionable.

An implementational issue is that the lack of topology preservation would make the neighbourhood Q-learning problematic in an ART network. It is possible that Q-updates could teach neighbouring units in input space rather than the topological space of the SOM, but this creates a challenge for parallelisation. Notwithstanding this, the ART network may be considered as a possible alternative to the SOM for representing and generalising over the input space.

Growing Neural Gas

The Growing Neural Gas (GNG) algorithm (Fritzke, 1995) dynamically generates both units and their topological structure in the data space in a way that does not require either the number of units or the dimensionality of the latent space to be specified a-priori. Units are both created and destroyed as the unsupervised learning progresses, allowing the tracking of a non-stationary environment. Another advantage is that there are few parameters to set. However, as with the ART network, the desired resolution of the representation is fixed beforehand which again means that the representation does not model the density of the data.

GNG is considered a promising alternative to the SOM for mapping the state space, especially given the provision of a flexible dimension neighbourhood for use by neighbourhood Q-learning.

Hamming distance

Mahadevan and Connell (1991) use two clustering techniques for generalising over the state space of a reinforcement learning problem. The simplest utilises a Hamming distance measure between binary vectors. For an action, a, taken in state, s, $Q(s,a)$ is updated towards the received reward, and so is $Q(s',a)$ for all s' within a fixed Hamming distance of s. This idea is analogous to the neighbourhood Q-updates of the Kohonen map, except that states are considered similar if they are close in input space rather than the physical space of the SOM. In Mahadevan and Connell (1991) each state is still represented explicitly which required them to perform some pre-processing dimensionality reduction on the sensory data for practical purposes. In this sense, online generalisation is not directly addressed.

Ad hoc statistical clustering

The second approach adopted by Mahadevan and Connell (1991) is more sophisticated and involves partitioning the state space into a number of *clusters* with each cluster maintaining a single Q-value (see 4.2 for a description of the algorithm). Although the approach is somewhat ad hoc in the sense that no formal or empirical analysis exists, it does have one novel feature of interest. The state space is partitioned differently for each action, thus removing the assumption that a particular classification of the input space will be appropriate for all actions. The approach is somewhat similar to an ART network, since clusters can be created if no existing cluster suffices, and existing clusters can be moved through the state space in response to new inputs. The same arguments regarding the balance of class construction and destruction (clusters are never destroyed) are applicable as to the ART network. As with ART, this technique could be used in the place of the SOM, and indeed has been shown by Mahadevan and Connell to be effective on a reinforcement learning problem similar to those considered in chapter 5.

Decision trees

A similar approach is that of decision trees in which an initially small number of regions covering the entire input space are iteratively split into smaller and smaller partitions until each region behaves consistently with respect to how the reward varies under different actions (see Chapman and Kaelbling (1991) for an example). This method benefits from being simple and intuitive, but is not ideally suited to interactive applications involving non-stationary environments because the splitting is something of a one way process. As we have already seen, the creation of new categories is generally the easy part, with the practical considerations of merging, adapting and destroying (to maintain an efficient representation) often overlooked.

Generative Topographic Map

The generative topographic map (GTM) (Bishop et al., 1998) offers a principled alternative to the SOM in which a faithful probability distribution is achieved and convergence to a local maxima of an objective function guaranteed. Additionally, the neighbourhood-preserving nature of the GTM is a guaranteed automatic consequence of the algorithm, while the conditions under which the SOM self-organises have not been quantified. In its original form, the GTM is a batch algorithm, although it could, in principle, be re-stated in an on-line form. The complexity of the algorithm is the same as the SOM, but the underlying mathematics is more complicated, and this is relevant for the conversion from the batch to the online version of the algorithm. Whereas the Kohonen network maps a point in the input space to a network unit which can then be used directly as a state in the reinforcement learning problem, the GTM operates the other way around, mapping a continuous two dimensional latent space (corresponding to a two dimensional SOM) to the continuous, higher dimensional input space. Hence additional processing (a search problem) is required to run the algorithm backwards so that a point in the input space can be mapped to a point in the latent space (Williams, 2000).

Elastic net

The elastic net of Durbin and Willshaw (1987) has already been mentioned as a principled alternative to the SOM, and would be well suited to the kind of representational problems considered here.

Direct parameterised approximation of the Q-function

The TD-Gammon application of Tesauro (1994) has already been introduced as an application of a feedforward network to the direct representation of the value function. The key difficulty here is that in representing the value function, $V(s)$, an explicit environment model is required in order to know which actions will lead to the best states. The QCON model of Lin (1993) addresses this by representing the Q-function (attaching values to state-action pairs) which involves maintaining a separate network for each action. The problem here, noted in section 4.6, is that online generalisation of the action space is neglected.

However, the use of backpropagation for generalising over both the state and action spaces is viable providing an appropriate method for embedding this supervised learning technique within an RL framework can be found. This is investigated in subsequent chapters.

In the meantime it is noted that backpropagation may be expected to scale better than any of the approaches described above. This is because very different outputs can be generated by similar input vectors. As an example, TD-Gammon encoded the number of pieces currently off the board with one input unit out of a total of 200 input units. Those familiar with backgammon will recognise that this knowledge will tend to have a large impact on the optimal strategy, yet the SOM is unlikely to be able to learn the special significance of this particular dimension of the input space in the face of 199 other weight comparisons. The distance measure used in the SOM gives equal influence to each input line making the classification very coarse, particularly in large dimensional spaces.

Justification of the SOM for input space representation

There are many attractive features of the SOM that make it desirable when compared to a variety of alternatives. There are drawbacks associated with the SOM too, but it has been argued that none of these represent a pathological problem in the context of the kind of reinforcement learning problems considered so far. In short, the SOM is a simple, robust and incrementally adaptive technique with a short processing cycle and intuitive visualisation. There is a wealth of empirical support for its effectiveness, it is easily run interactively, can be used and trained simultaneously, and efficiently provides explicit representations on which to base empirical analysis and additional layers of processing. It balances construction and destruction of categories, supports Q-neighbourhood learning and lends itself to parallelisation. However it is noted that a number of other alternative techniques could be used to represent the input space, including some not mentioned above.

6.2.8 Assumptions of the Input map

It is now possible to draw up the assumptions under which the SOM will be successful in the context of the proposed model.

❶ We require that the Input map produces a stable, approximate distribution, at a sufficient resolution, of the situations that the agent encounters. The first assumption is therefore that a set of parameters can be found that allows this to happen. Of particular importance is that the network is of an appropriate size and dimension and that it becomes stable while there is still enough plasticity in the Q-learning algorithm to perfect the estimates of those stable states.

❷ Unfortunately the input distribution will not generally be stationary but will change as the rest of the system changes. This may be the result of changes in the environment, but there are other forces in play too. For example, the mapping of input space will directly affect the Q-values associated with those states and therefore also the actions that are learned. This will then affect the class of situations encountered by the agent, and therefore the input to the map, creating a cycle of consequence.

We would like to know that the input distribution will change only slowly with respect to *all the free parameters of the whole system (including the Motor map, the Q-values, and the environment itself)*, and converge to a static distribution, since a static input distribution is a usual assumption of the SOM. The second assumption is therefore that the input distribution converges to a stationary distribution.

❸ The third assumption is that the same classification of input space will be efficient and appropriate for all actions. In the case of Mahadevan and Connell (1991), separate clusters, or classes were maintained for each action. In the approach described here, a single classification scheme must serve all actions.

6.3 Q-learning

The Q-learning module of the proposed system is now analysed.

It has already been stated that the RL technique adopted here represents something of a minimal RL algorithm in the sense that the expected return of equation (2.2) is estimated as simply as possible. Specifically, a finite-horizon Monte Carlo approach is adopted which assumes that all reward for a given action will be received within a small, fixed time window following that action. The justification for this minimal approach is based on the fact that the focus of this thesis is not the value estimation technique itself but rather the techniques for generalisation.

In principle, we expect to be able substitute Q-learning for the current value estimation method without loss of generality, allowing full support for arbitrarily discounted rewards. An increase in performance may also be achieved, particularly if one of the augmented versions of the algorithm such as *DYNA-Q* (Sutton, 1990) were used[3]. Ad-

[3]This approach accelerates the passing of reward information around the Q-table by approximating an underlying state-transition model and then using this model to update Q-values that are likely to influence each other. Recall that in standard Q-learning, return estimates depend heavily on other return estimates (hence *bootstrapping*).

ditionally, following the suggestions of Sutton (1996) that $0 < \lambda < 1$ may be preferred[4], the use of eligibility traces (see section 2.8) is also suggested. These more sophisticated algorithms expedite the passing of reward information around the Q-table, and are strongly recommended for future work involving scaled versions of the proposed model in which delayed rewards play a significant role. However, it is reiterated that the current issue is one of *structural* credit assignment rather than *temporal* credit assignment. i.e. we are interested in linking states on the basis of their state space coincidence rather than their temporal coincidence.

With this discussion in mind, the analysis of the 'Q-learning' subsystem of the proposed model will be content to appeal to the standard Q-learning results in the expectation that they will be relevant enough to illuminate the assumptions underpinning the success of the proposed model.

6.3.1 Assumptions

Q-learning has been analysed in detail (Watkins, 1989; Watkins and Dayan, 1992) and the policy it generates proved to asymptotically converge to an optimal policy under the following assumptions:

❶ The task is formulated as a Markov Decision Process.

❷ The evaluation function is a lookup table.

❸ Each state-action pair is tried infinitely often.

❹ An appropriate learning and exploration rate is used.

These assumptions have already been discussed in section 2.9, in which it was discovered that in practice few or even none of these criteria guaranteeing success can generally be satisfied. In particular, it was noted that the MDP assumption is violated by perceptual aliasing problems and continuous state and action spaces. Clearly both

[4] Recall from chapter 2 that this parameter controls the degree to which actual reward values and other Q-values are blended to produce the target estimate.

apply to the experiments of chapter 5. The proposed model does succeed in satisfying the look-up table criteria, although the states and actions of this table are moving targets, unlike in standard Q-learning. That each state-action pair is continually visited is clearly left unsatisfied by the inevitability of finite length trials, although the moral behind the assumption that care should be taken when setting the amount of exploration in the system, can be respected. According to stochastic approximation theory, the fourth assumption can be satisfied by the criteria of equation (2.14). Note that the first condition ensures that there is always enough plasticity to overcome any adverse initial conditions, and the second requirement guarantees eventual stability (by guaranteeing finite variance), given an infinite trial length. In practice, given finite trial lengths, other learning rates, such as the one used in chapter 5 may be empirically judged to be satisfactory.

The key question is to what extent the violation of the first two assumptions in particular, as a result of the need to generalise, will compromise convergence to optimality. Results such as those presented in Tesauro's *TD-Gammon* system appear to suggest that as long as appropriate generalisation is used, good results can still be expected. These findings are supported in the literature (Crites and Barto, 1996; Mahadevan and Connell, 1991; Ziemke, 1996) and also in the preliminary experiments of this thesis.

6.3.2 Interaction between the Input map and Q-learning

We know that for the Q-learning part of the system to be successful, the environment and the reward function should be stationary. In the case where we expect either to vary we must be sure to maintain sufficient plasticity in the Q-learning part of the algorithm, at the cost of convergence. But even if the environment is stationary, the states (and actions) of the Q-table will change as learning in the other parts of the system takes place.

Ideally the Input map would be allowed to form and stabilize, after which the Q-learning algorithm could be trusted to find suitable Q-values from each of these static states. However, it has already been noted that this ideal is unrealistic, since there is interaction between the Q-values and the input distribution in both directions. Therefore in practice it is necessary to anneal the plasticity of the Input map and the Q-learning process *simultaneously*.

Recall that the covariance learning algorithm of Wedel and Polani (1996) (see section 4.5) also used a SOM to map the input space, with each unit then learning a 'best action' by trial and error sampling of the reward function. They concluded that running the SOM concurrently with the rest of the learning system was inappropriate, and they settled on beginning with a phase of mapping the input space, followed by a phase of learning Q-values and actions. This cycle was then iterated to address the interactions between the different parts of the system. Although their approach was different to the model proposed here — it involved approximating the reward function with a Normal distribution, and each unit of the Input map only maintained a single 'private' action with no concept of shared action units — it remains unclear as to why the same problems were not encountered in experiments performed as part of this thesis.

It is clearly important to select parameters carefully so that the interaction between the Input map and the Q-learning is sensible. In particular, small changes to the Input map may entail significant re-estimation of Q-values, whereas in most cases we expect even large changes to the Q-values to yield only modest changes in the input distribution. Hence the Input map should probably change slowly with respect to the rest of the system. As an example, consider what happens when a unit moves across the diagonal line in the state-space diagram of figure 5.19. That unit must learn to turn in the opposite direction which involves exploring the alternative actions and significant re-estimation of the relevant Q-values. This process will take considerably longer than it took that single unit to make the short transition from one half of the input space to the other.

Based on this discussion, there are two guidelines for setting the parameters of these two sub-systems. Firstly, plasticity in both the Input map and the Q-learning process should be maintained in parallel. Secondly, the plasticity in the Input map should generally be small compared with that in the other parts of the system. A neat solution might involve linking one to the other in some way, and then linking both to the changes in the environment and reward signal. Now each subsystem would be capable of reacting to changes in its own specific environment, which would of course include the other subsystems. However, this is significantly more complicated than the time dependent annealing approach adopted in chapter 5 and is not explored further here.

We conclude by noting the implicit assumption that stable dynamics can be achieved by an appropriate set of parameters. More work needs to be done to determine when stability is likely to be compromised as a result of interaction between the Input map and the value-estimation modules of the system.

6.4 Learning actions

Our attention now turns to the Output map and its ability to generate an appropriate set of actions for use by the rest of the system. Assuming that a suitable set of actions *are* provided, then the previous sections lead us to expect the rest of the system to do a good job of maximising reward. Therefore we expect the success of the architecture to hinge on the efficient and effective operation of the Output map. In this section, we ask whether there are some reward functions which can be learned more easily than others, and how long we can expect to wait before suitable actions are discovered. This last point is more than a performance issue since we need this estimate to guide the parameters of the rest of the system. For example, plasticity in the Q-learning subsystem must be maintained until an appropriate set of actions have been discovered. Other issues are also considered including pathological limitations, efficiency, alternative learning rules, complexity, finding parameters, and scalability. Finally, a comparison of the proposed model with existing alternative approaches for representing and generalising over the action space is offered.

In the following analysis, a number of simplifications are made. We consider first an Input map consisting of just a single unit, u_1. Since this unit must always 'win' for any situation, the position of the unit in input space is irrelevant. This single Input unit is now connected to a single Output unit, a_1, which represents a degenerate Output map. Every input stimulus is identical, and the sole task of this simplified system is to discover the optimal location within the output space for a_1 given a reward function, r, in the range $[0,1]$. For consistency with the previous experiments, and to aid visualisation, the output space is two-dimensional, and for simplicity a reinforcement learning horizon of one is used so that reward information is immediately available after taking an action and no discount factor need be considered. The usual algorithm is used, with the relevant simplifications. At time t:

❶ Take Action, $\langle M_L, M_R \rangle = \langle w_1 + MA \times \text{random(-1,1)}, w_2 + MA \times \text{random(-1,1)} \rangle$.

where $\langle w_1, w_2 \rangle$ are the weights of the single action unit a_1.

❷ Calculate the reward, R, of taking action $\langle M_L, M_R \rangle$.

❸ If $R > Q(u_1, a_1)$ then update a_1 towards $\langle M_L, M_R \rangle$ proportional to the learning rate, OL.

i.e.
$w_1 = w_1 + OL \times (M_L - w_1)$
$w_2 = w_2 + OL \times (M_R - w_2)$

❹ Update $Q(u_1, a_1)$ towards R by a factor of $\alpha < 1$.

i.e.
$Q(u_1, a_1) = Q(u_1, a_1) + \alpha \Big\{ R - Q(u_1, a_1) \Big\}$

❺ Return to ❶.

To summarise, at each time-step a perturbed action is taken and the resulting reward used to update the single Q-value of the system. If and only if the reward is greater than the existing Q-value, then the action represented by a_1 is moved towards the perturbed action. Notice that there are three main parameters, MA, OL and α. MA controls how much noise is added to the proposed action of a_1. If $MA = 1$, then a completely random action is selected with equal probability, providing we turn a blind eye to edge-effects[5]. For small MA only small perturbations are made to the proposed action, and if $MA = 0$ a_1 will be stationary in the output space because there is no exploration. The second parameter, OL, is the factor by which the weights of a_1 are updated towards the perturbed action. It might appear that $OL = 1$ would be best since

[5]By 'edge effects' we mean the consequence of truncating the components of exploratory actions which lie outside of the range $[0, 1]$.

any discovery that offers improved reward can then be instantly learned. However, the reward signal may be noisy, or follow a probabilistic distribution in which the same action could yield different rewards on different occasions. The learned action must optimise reward over the stochastics of the environment and not be too easily misled by individual samples. In general then, *OL* should be small. The final parameter, α, is the learning rate of the Q-learning algorithm which, for similar reasons, should generally be small.

Under the simplifications described above, the reward function, r, can be thought of as a function of two variables corresponding to the two dimensions of the output space. To simplify matters further, let us assume that the reward function is stationary and deterministic so that $r(x,y)$ always yields the same value for the same x and y. Now a simple reward surface can be plotted over the output space as in figure 6.3. In this example, $r(x,y)$ is negatively proportional to the distance of the point $\langle x,y\rangle$ from the point $\langle 0.5,0.5\rangle$, creating a single maximum at $\langle 0.5,0.5\rangle$.

Using small *MA*, *OL* and α, a typical trajectory of unit a_1 through the output space and across the reward surface is shown in figure 6.3. Although the maximum is always found, notice that the path need not be direct. To illustrate this, consider a particular position and an accurate Q-value of a_1 at some time t. Let us call these values $\langle x_t,y_t\rangle$ and Q_t respectively. A subsequent sample, $\langle x_{t+1},y_{t+1}\rangle$, of the space around $\langle x_t,y_t\rangle$, which happens to yield a reward less than Q_t, will not affect the position of a_1, although the Q-value of a_1 will go down as a result of this 'unlucky' sample. Next, a point in output space, $\langle x_{t+2},y_{t+2}\rangle$, could be sampled which yields a higher reward than Q_{t+1}, even though $r(x_{t+2},y_{t+2}) < r(x_t,y_t)$, because Q_{t+1} is no longer a precise estimate of $r(x_t,y_t)$. The result is that the position of a_1 is updated towards $\langle x_{t+2},y_{t+2}\rangle$ and to a *worse* position on the reward surface than $\langle x_t,y_t\rangle$. In other words the trajectory of a_1 is influenced by noise generated by the exploratory process. However, as the three parameters, *MA*, *OL* and α are annealed, a_1 is observed to converge to the maximum on every trial.

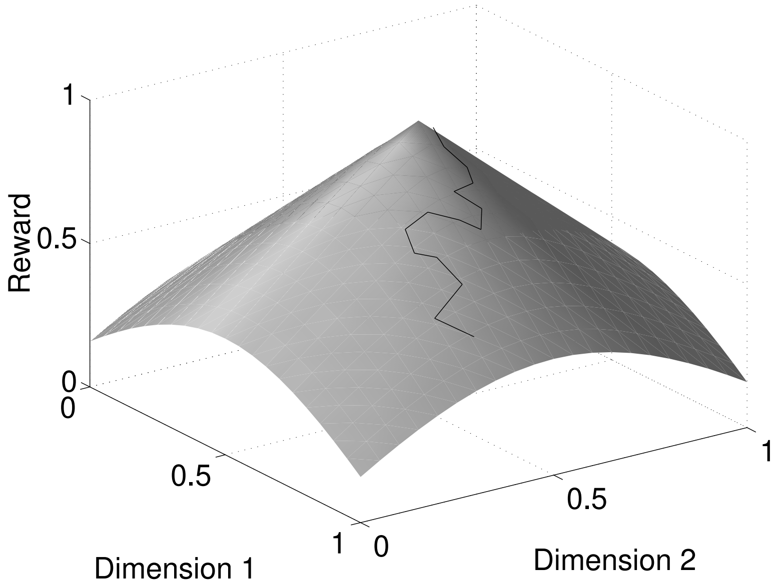


Figure 6.3: Learning on a basic reward surface.

6.4.1 Local maxima and attractor forces

We can see that when MA, OL and α are small, learning approximates a gradient ascent algorithm. However, since a sampling method is used, the gradient need not be known explicitly and no assumptions regarding the reward function such as its differentiability need be made. But in any gradient ascent method, local maxima are a problem, and this is illustrated in figure 6.4. The end position of a_1 is shown for a hundred trials, with each trial starting in a random position. On each occasion, a_1 simply climbs to the top of the hill that it starts on.

The maxima get higher towards the middle of the space, and it is obviously desirable that a_1 can discover and exploit this fact. The simple answer is to increase the exploration range of a_1 so that it can sample more regions of the space and discover better maxima. Fixing $MA = 1$ means that all points in the output space are sampled with equal probability. As long as OL and α remain small, a_1 will move towards regions of highest reward. The new distribution of final positions of a_1 on the same landscape after a similar number of runs with these new parameters is shown in figure 6.5. Now the units no longer sit on local maxima, but at least occupy the region of highest reward.

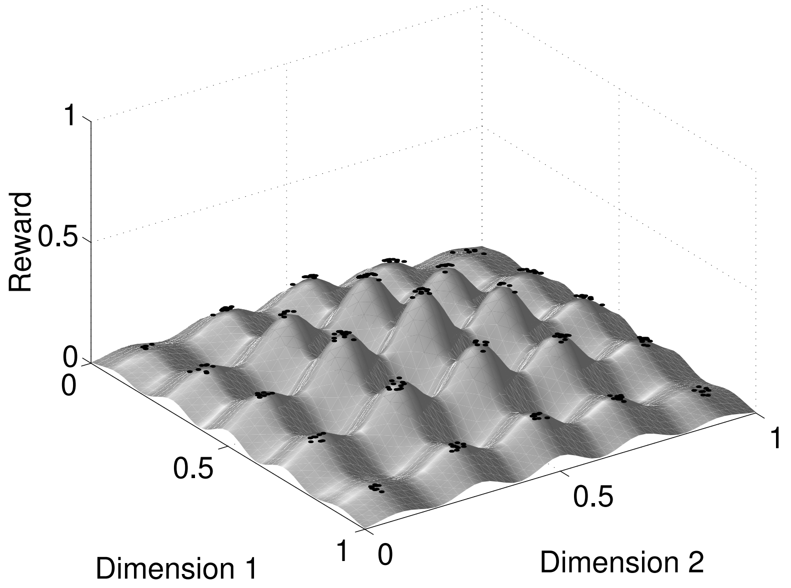


Figure 6.4: A reward surface containing local maxima of varying heights. The final position of a_1 in each of 100 trials is denoted by a black dot. *MA*, *OL* and α are all fixed and small (set to 0.2), and the algorithm approximates a hill-climb.

Since all points of the output space are sampled with equal probability when $MA = 1$, the Q-value of a_1 will approximate the mean value of $r(x,y)$ over the entire space. This approximation will become more accurate (but take longer to acquire) for smaller fixed α. Since the position of a_1 in the output space is only updated towards a sampled point $\langle x,y\rangle$ when $r(x,y) > Q(a_1)$, it follows that each point $\langle a,b\rangle$ for which $r(a,b) > Mean(r)$ exerts an attractive force on a_1 proportional to the distance between $\langle a,b\rangle$ and the position of a_1 (because of the update rule ❸ above).

In general, the position $\langle w1,w2\rangle$ of a_1 will be stable only if these attractive forces cancel, i.e. when the following two conditions are met:

$$\int_{w1-MA}^{w1+MA} dx \int_{w2-MA}^{w2+MA} dy\ (x-w1) \times G(x,y) = 0$$
$$\int_{w1-MA}^{w1+MA} dx \int_{w2-MA}^{w2+MA} dy\ (y-w2) \times G(x,y) = 0 \tag{6.3}$$

$$\text{with } G(x,y) = \begin{cases} 1 & \text{If } r(x,y) > Mean(r) \\ 0 & \text{Otherwise} \end{cases} \qquad Mean(r) = \frac{\int_{w1-MA}^{w1+MA} dx \int_{w2-MA}^{w2+MA} dy\ r(x,y)}{(2MA)^2}$$

Since these equations are satisfied only by the point $\langle 0.5, 0.5 \rangle$ in the surface of figure 6.4 when $MA = 1$, we can expect a_1 to be stable at $\langle 0.5, 0.5 \rangle$ in the limit of an infinite sample size providing OL and α are small and (figure 6.5).[6] But when $MA < 1$, then the equations can be satisfied by sub-optimal maxima, providing MA is less than the radius of the maxima. By starting with $MA = 1$ and annealing to $MA = 0$, a gradual transition can be made from locating a large region with reward above $Mean(r)$, to climbing to the local summit of this region. Figure 6.6 shows the effect of annealing MA in this way according to the usual schedule, with $f(t) = 0.9995^t$. The annealing schedule itself is not emphasised, and no attempt was made to optimise this schedule in terms of minimising learning time. The figures of 6.5 and 6.6 show the process of convergence for $\alpha = OL = 0.2$. The process can empirically be shown to converge in the limit by decreasing these parameters further. Figures 6.7 and 6.8 show the same plots but for $\alpha = OL = 0.01$ in which convergence is much better. Only twenty runs are shown in each plot, and in figure 6.8, the reward surface is offset slightly to show that the model is actually seeking the peaks, and is not just coincidentally attracted to the centre point of the space.

[6] We also hope for convergence to this point since when $w1 < 0.5$, the left hand side of the top equation in (6.3) appears to yield a positive value, and when $w1 > 0.5$, it appears to yield a negative value. The same is true for $w2$ and the lower equation. However, this is not proved for the general case.

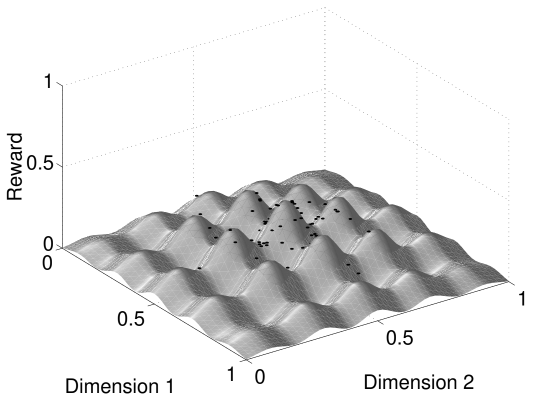


Figure 6.5: Using a large exploration parameter, $MA = 1$, and small, fixed learning rates, $OL = \alpha = 0.2$.

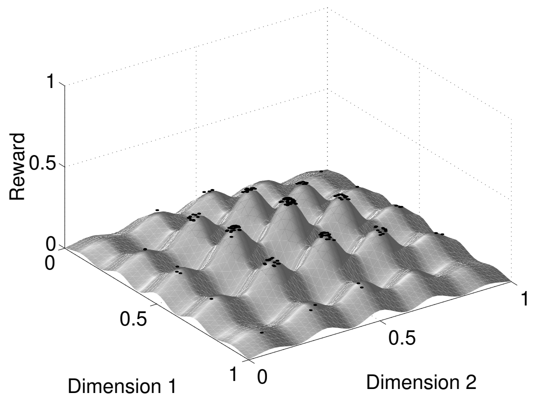


Figure 6.6: Annealing MA from 1 to 0, with small OL and α. a_1 is preferentially drawn to the highest peaks.

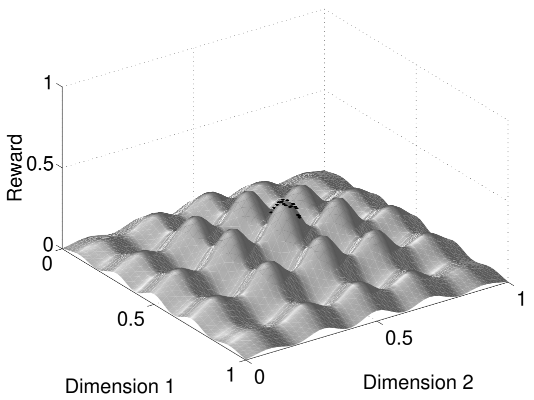


Figure 6.7: Using a large exploration parameter, $MA = 1$, and even smaller, fixed learning rates, $OL = \alpha = 0.01$.

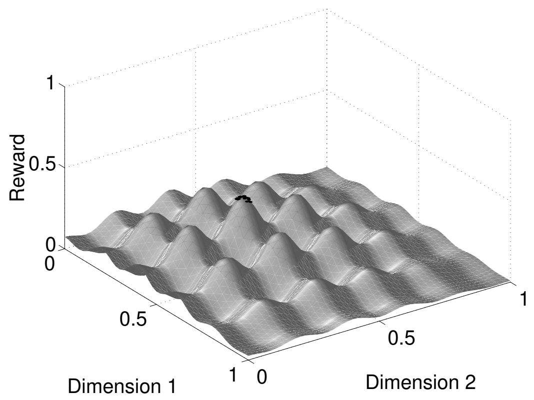


Figure 6.8: Annealing MA from 1 to 0, with $OL = \alpha = 0.01$. a_1 is preferentially drawn to the highest peak.

Figure 6.9 shows the effect annealing MA has on the reward received over time for the landscape of figure 6.6. The sinusoidal signature is interesting because it illustrates the influence of the exploration radius on the reward. As MA is reduced, the sample region covers a different number of peaks and troughs of the reward surface. Hence $Mean(r)$ within the exploration region has a sinusoidal property as that region is reduced in size, resulting in the type of curve shown in the figure. At $t \approx 5000$ the exploration region

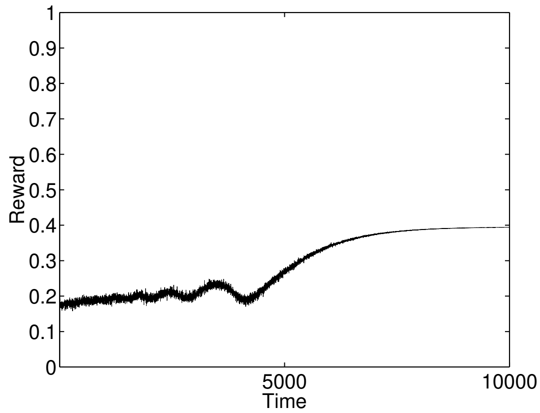


Figure 6.9: The effect on the reward of annealing *MA* ($MA = 0.9995^t$), averaged over 50 trials. Unlike previous reward graphs, *every* time-step is plotted — i.e. there is no averaging inside each trial.

is the radius of the main peak in the centre of the space ($MA = 0.1$), and as this peak is climbed and *MA* annealed further, the reward asymptotically converges to 0.4, which is the height of the central peak.

It may seem that this learning scheme has the potential for a_1 to be caught between one or more regions of high reward, and to find none of the maxima as it is dragged by forces which essentially cancel each other out. In fact this does not tend to happen, because as the exploration becomes smaller, all attractors other than one will gradually fall out of reach of the sampling region. For example, if a_1 is started in the centre of a regular basin, it will initially be attracted in all directions and will hence sit in the middle. But as *MA* is reduced, one direction will be favoured (at random, if there are no better criteria), and a hill climb will begin. An illustration of this kind of effect is considered shortly.

6.4.2 Pathological behaviour

However, problematic reward functions do exist, and figure 6.10 demonstrates one such function. Because a_1 is essentially drawn towards an attractor region with a strength depending on the cross-sectional area of the reward function above $Mean(r)$ for the current exploration region, we expect a_1 to locate regions with *large* areas

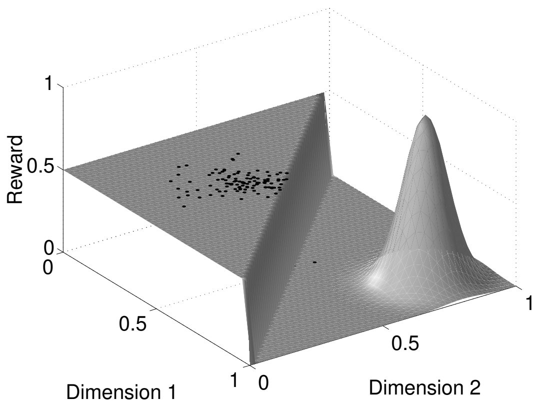


Figure 6.10: Deceptive reward functions. The large size of the plateau provides a greater attractive force than the peak, even though the latter is higher.

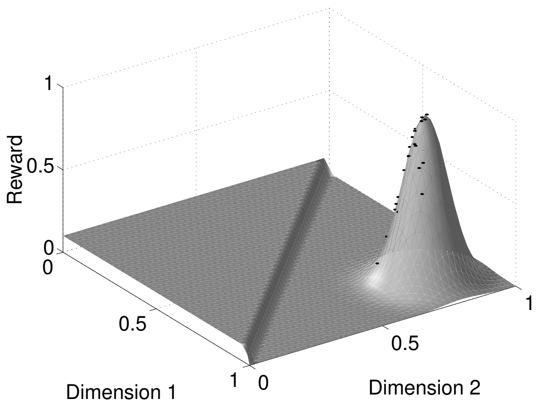


Figure 6.11: Deceptive reward functions. The height of the plateau is less than $Mean(r)$ and so provides no attractive force.

above $Mean(r)$ rather than areas with *high* r. In figure 6.10 it can be seen that the large plateau on the left of the surface presents a much greater attractive force than the peak on the right, even though the peak is higher, and so the units are drawn to the plateau. Only when the height of the plateau is reduced to below $Mean(r)$ for the whole surface, is the attraction force of the plateau eliminated, and a_1 is drawn towards the peak (See figure 6.11).

Although in general we would not expect to be able to integrate the function $G(x,y)$, equation (6.3) still provides an insight into the expected behaviour of a_1. Given the discontinuous reward surface of figure 6.12 for example, we can immediately see that the cross-sectional area above $Mean(r)$[7] is twice as large on the left of the output space as on the right. If $d1$ is the distance between a_1 and the maxima in the $\langle 0,0\rangle$ corner of the space, and $d2$ is the distance between a_1 and the maximum in the opposite corner, equation (6.3) gives the approximate equality:

$$d1 \times \Big(Area_{left} > Mean(r)\Big) = d2 \times \Big(Area_{right} > Mean(r)\Big) \qquad (6.4)$$

[7] $Mean(r)$ for the whole output space that is. This implies $MA = 1$ of course.

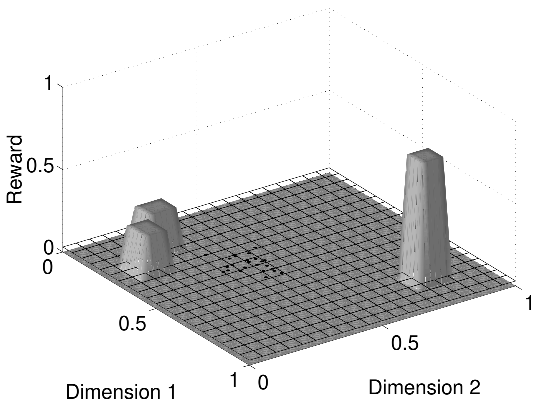


Figure 6.12: MA = 1. a_1 is drawn to an equilibrium point that satisfies equation (6.3). The grid represents the $Mean(r)$ for the whole space.

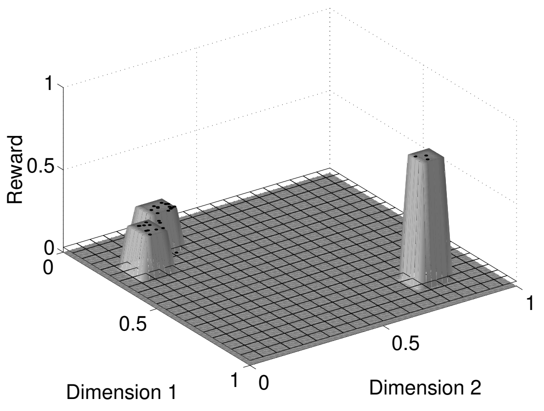


Figure 6.13: MA is reduced to zero. a_1 is more likely to be dragged to the maxima at the $\langle 0,0 \rangle$ corner, because these have a larger area above $Mean(r)$.

from which $\frac{d1}{d2} = \frac{1}{2}$. Figure 6.12 shows the distribution of final positions of a_1 after learning for fixed $MA = 1$, and the mean of this distribution appears to satisfy equation (6.4). If MA is now annealed to zero over the course of the experiment, the final distribution is as shown in figure 6.13. Notice that a_1 is drawn to the weaker attractor with a non-zero probability. This probability is related to the variance of the distribution of a_1 at the critical value of MA where one of the attractors falls out of sight of the exploratory process. In turn, this variance is dependent on the parameter OL. The smaller OL, the smaller the variance, the lower the probability of a_1 ending up on the right hand maximum (the weaker attractor, despite its height), but the longer the learning takes. For this experiment, small fixed learning rates, $OL = \alpha = 0.2$ were again used. All Q-values were initialised to zero and updated thereafter using step ❹ on page 174. Note that while this biases the estimate of $Mean(r)$, convergence to the true value of $Mean(r)$ is still expected in the limit, given learning rates that satisfy the usual convergence criteria of 2.14.

6.4.3 Stochastic hill-climbing

This analysis shows that the learning algorithm does not perform the usual kind of hill-climbing search. A number of differences are apparent. In hill-climbing:

- The *value* of the current set of parameters (being optimised) is known precisely, rather than through an incremental and noisy estimate.
- The parameters are only changed when a sampled value exceeds the *actual* value of the current parameters rather than the *estimated* current value.
- When this happens, the current set of parameters are *replaced* by the new set, rather than just updated towards the new set.

In other words, the learning algorithm presented here turns every feature of a hill-climbing search into a stochastic process. This has to be the case because the properties of the reward surface may only be defined stochastically, and any single sample may be unrepresentative when considered on its own. There is also assumed to be noise in the system, so the learning algorithm must not over accommodate each sample, and cannot afford to take any particular piece of information too literally. A consequence of these differences is that while a hill-climber will tend to find high peaks, the learning algorithm presented here will tend to find the largest 'better than average peaks'. In theory, a_1 could end up at an arbitrarily low point on the reward surface. This could apparently be engineered by recursing the process seen in figures 6.12 and 6.13 where low wide peaks are favoured over tall thin ones. This would lead to a very irregular reward surface.

The merits of being attracted preferentially to large 'good' regions rather than small optimal regions may have positive implications for reliability — particularly under the influence of the Kohonen mapping of the output space in which units will be dragged around by their neighbours. While it would seem to be possible to engineer a reward surface that results in arbitrarily poor performance, in practice, for slowly and smoothly varying reward functions, performance seems to be good. Consider for example figure 6.14 in which one might expect point A to be favoured because this is initially (i.e. for

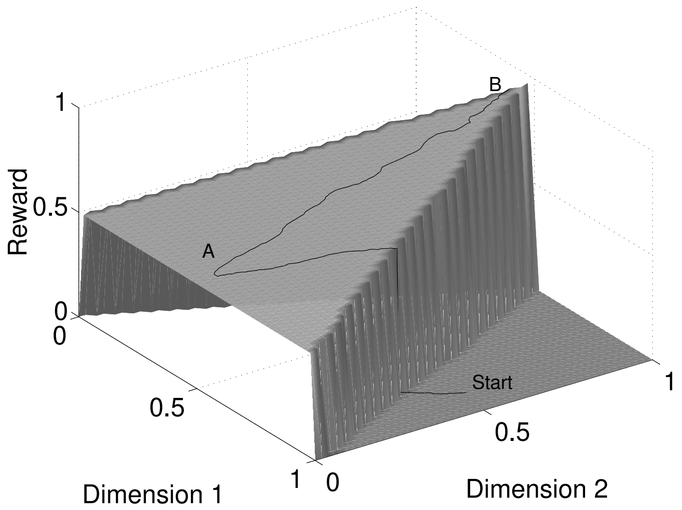


Figure 6.14: Path of a_1 during learning. Initially point A is considered optimal and a_1 moves towards it. As *MA* becomes smaller, a hill-climb to point B is performed.

large *MA*) where the attractive forces cancel (taking into consideration edge-effects). At first, for large *MA*, a_1 is indeed attracted to this point at the bottom of the plateau, but as *MA* is reduced, a hill-climb to point B is performed. The hill-climb starts only when *MA* is reduced to the point at which a sufficiently local region is sampled which allows discrimination between points on the plateau.

6.4.4 An alternative learning rule

The current learning rule for updating the position of a_1 is:

$$w_1 = w_1 + OL \times (M_L - w_1)$$
$$w_2 = w_2 + OL \times (M_R - w_2)$$

An interesting alteration to the learning rule is to add a factor representing the height of the sample:

$$w_1 = w_1 + OL \times R \times (M_L - w_1)$$
$$w_2 = w_2 + OL \times R \times (M_R - w_2)$$

where R is the reward of taking action $\langle M_L, M_R \rangle$. This changes the stability equalities to:

$$\int_{w1-MA}^{w1+MA} dx \int_{w2-MA}^{w2+MA} dy \; (x - w1) \times r(x,y) \times G(x,y) = 0$$
$$\int_{w1-MA}^{w1+MA} dx \int_{w2-MA}^{w2+MA} dy \; (y - w2) \times r(x,y) \times G(x,y) = 0 \tag{6.5}$$

and the behaviour changes predictably, with the *volume* of regions above $Mean(r)$ now providing the attractive force. In figure 6.15 the single peak is three times higher than the other two peaks, so the ratio of attraction forces is 3:2 in favour of this peak, and hence the learning algorithm preferentially finds this peak. But in figure 6.16, the height of the single peak is reduced to double that of the other two, resulting in equal attraction forces with the predictable consequence that a_1 discovers each region with equal probability. There is now a trade-off between height and width of the reward surface, and the attraction force of a region can be thought of as proportional to the *volume* of the reward surface above $Mean(r)$ instead of the cross-sectional area.

$G(x,y)$ could be removed from the stability equalities because the height of the reward surface is now taken into consideration by the $r(x,y)$ term. This would correspond to always updating the position of a_1 rather than just when $R > Q(a_1)$, and gives the learning rule a more consistent form. However, this is not investigated here.

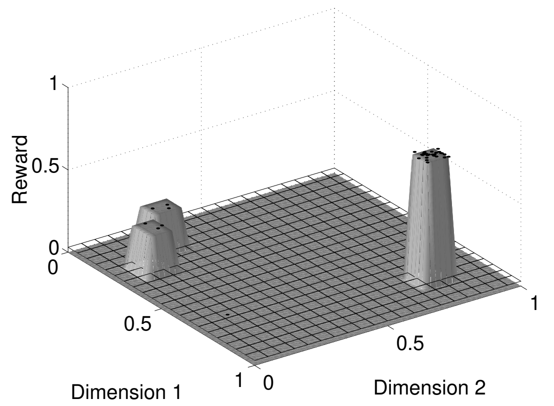


Figure 6.15: Learning based on the volume of the reward surface above $Mean(r)$. Now the higher peak (three times higher than the other two peaks) provides an attractive force 1.5 times as strong as the two smaller peaks.

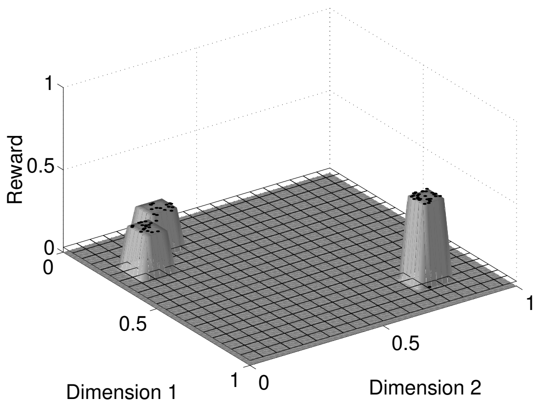


Figure 6.16: The same experiment with equal volumes above $Mean(r)$.

6.4.5 Summary of the behaviour of a single unit

So far the general behaviour of a single Output unit in a two-dimensional space has been discussed. The unit is attracted towards points of equilibrium where two orthogonal sets of attractive forces cancel. When the parameter *MA* is large, the unit is *generally* attracted to either large regions of the reward surface above $Mean(r)$ or voluminous regions of the reward surface above $Mean(r)$, depending on which incarnation of the learning rule is used. As *MA* is annealed, a hill-climb to the top of the local feature is performed. The algorithm does not need a differentiable reward function, and local minima do not present a problem until *MA* is smaller than the undulations in the landscape. However the main drawbacks are identified as:

- There is no guarantee that regions of high reward will be found if lower regions have larger areas or volumes above $Mean(r)$.
- There is not even a guarantee that regions with large areas or volumes above $Mean(r)$ will be found. A malicious reward surface could ‘hide’ such regions within larger regions of low area or volume.

- The exploration is generated by random noise which is a slow search method on its own. The more accurate the sampling algorithm must be, the smaller the parameters α and *OL* must be, and the longer learning will take.

6.4.6 Assumptions

Some assumptions underlying the efficient learning of actions can now be stated:

❶ Good positions on the reward surface are not hidden within larger regions of lower reward.

❷ A suitable set of parameters, *OL*, *MA* and α, can be found that allow sufficiently accurate sampling of the environment without sacrificing an acceptable learning time.

6.4.7 Scaling

There are a number of important differences between the simplified system discussed so far and the full architecture. In the full system:

- The output space could have any number of dimensions.
- There could be an arbitrary number of units in both the input and output spaces.
- Because of the self-organising property of the SOM, these units will influence the positions of their neighbours.
- Units learn from each others' Q-value updates.
- Any number of Input units can end up utilising any number of Output units. Some Output units may be redundant, or a single Output unit may end up serving multiple situations.

Assessing the behaviour of the Output map in the general case of the full system is difficult, and in some ways this difficulty is a drawback of the design. We already know the importance of finding appropriate parameters because of the interactions between the learning modules. Yet without a principled analysis, assumptions underlying useful operation and assertions pertaining to general performance are hard to come by.

A question of particular interest asks what kind of mappings between the input and output spaces are easy to learn, and which are hard. It would certainly be useful to have this information before specific applications are ventured. With respect to this question, continuity in the mapping must be a useful property. There is a general theme of units learning from neighbours and therefore mappings that are continuous — i.e. close regions in input space receive similar reward for actions close in output space — will be preferred. Figure 5.42 showed that the system is capable of learning non-continuous functions, but in general these will take longer to learn, and more care will be required in setting parameters. This is because discontinuous regions of the mapping can only be learned after the neighbourhoods are annealed to a point where these regions no longer significantly influence each other. This can be visualised by considering the reward functions as a surface over the combined dimensions of both the input and output spaces. The discussion suggests that the system will work best when this surface is smooth and varies only slowly.

Another consideration is the number of units in the Output map. If 90% of the situations the agent finds itself in require an action of type A and the other 10% require an action of type B, then under an assumption of faithful distribution, at least ten Output units will be required. In fact the faithful representation assumption is unsatisfied as we have already seen, and so the relationship between the likelihood of requiring an action and the number of units required to represent that action in the map is not so straight forward. In practice it may well be necessary for the designer to perform a small amount of off-line trial and error to search for a suitably sized Output map. As with the Input map, too many units is preferable to too few because an abundance of units can always learn from each other, or just become redundant, but an insufficient number of units will always degrade performance. The issue of scaling is considered more in chapter 8 where the proposed model is compared with one based on the back-propagation algorithm.

6.4.8 Increasing the dimensionality

It would be interesting to know how learning is affected by the dimensionality of the output space. Consider again the reward surface shown in Figure 6.17 where the reward at a point $\langle x, y \rangle$ is equal to $1 -$ the distance between $\langle x, y \rangle$ and $\langle 0.5, 0.5 \rangle$, scaled to fit the range $[1, 0]$. A similar reward surface can be generated for an output space of arbitrary dimension by setting the reward at a point $\langle x1, x2, x3, ..., x_n \rangle$ negatively proportional to the distance between that point and $\langle 0.5, 0.5, 0.5, ..., 0.5 \rangle$. The simplified architecture involving just a single Input unit and a single Output unit is now trained with this reward signal for a variety of dimensions and the results plotted. Figure 6.18 shows a number of plots of reward over time for dimensions ranging from one to four-hundred. Note that the optimum of $R = 1$ is never attained, even for the single dimension case, because the exploration in the system is fixed throughout ($MA = OL = \alpha = 0.2$). Note also that the average reward does not improve for the first few time-steps. This is because it takes some time for the Q-value estimate to become sufficiently accurate (through its random sampling and incremental update) for a_1 to start making sensible decisions as to where to move in the action space.

Figure 6.19 shows how the number of iterations required to reach a given average reward, R, varies with the number of dimensions. Two plots are shown, one for $R = 0.5$ and one for $R = 0.6$, corresponding to the two horizontal lines in figure 6.18. It can be seen that the relationship appears approximately linear, suggesting that the learning time increases linearly with the number of dimensions for this simple reward function.

This clearly represents something of a best-case scenario since the variables are decoupled — i.e. the function can be maximised by maximising the variables independently. Furthermore, for this particular example, we expect half the random explorations of the position of a_1 to yield an improvement in reward. In contrast, consider figure 6.20 in which only a small proportion of explorations will yield an improvement in reward once the action unit is on the ridge. By increasing the dimensionality of the space, and reducing the width of the ridge, the learning problem can be made arbitrarily hard for a noise driven hill-climbing process. This where some form of reward surface modelling such as that adopted by Wedel and Polani (1996) is expected to improve performance.

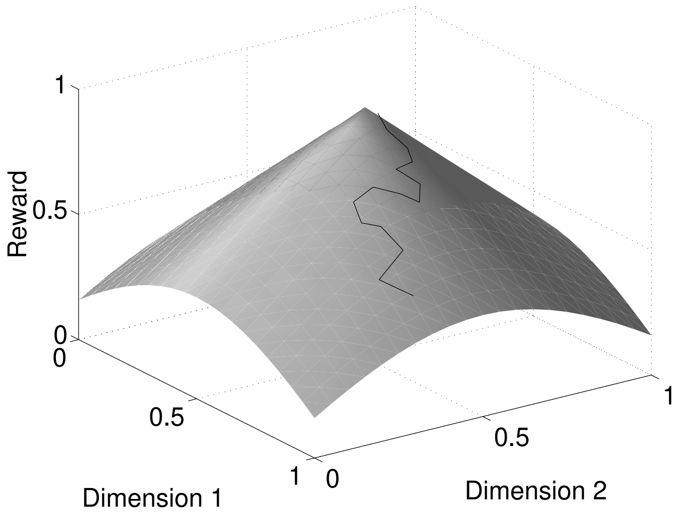


Figure 6.17: Basic reward surface. $r(x,y) = 1 - dist(x,y)$ where $dist(x,y)$ is the distance between $\langle x,y \rangle$ and $\langle 0.5, 0.5 \rangle$.

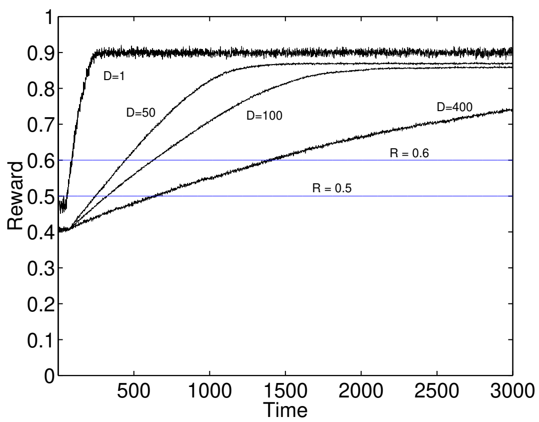


Figure 6.18: Plots of reward over time for output spaces of different dimensions (1,50,100 and 400). $MA = OL = \alpha = 0.2$. Each plot is averaged over a number of runs.

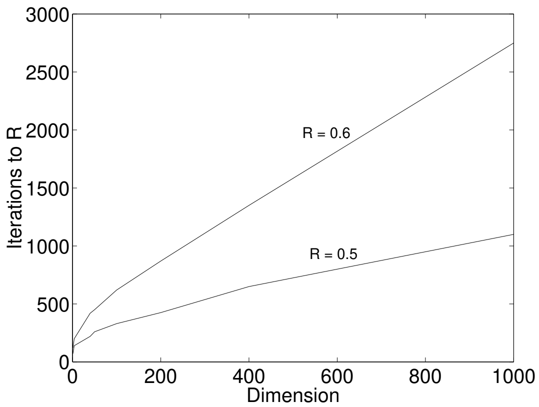


Figure 6.19: Graph shows how the number of iterations required to reach a given average reward, R, varies with the number of dimensions. Two plots are shown, one for $R = 0.5$ and one for $R = 0.6$.

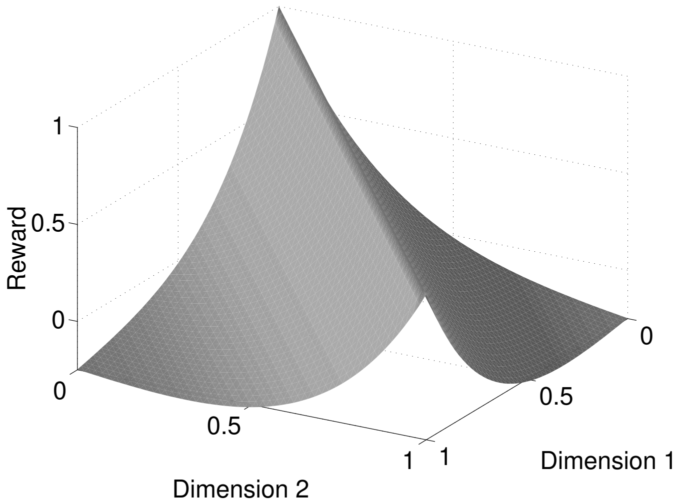


Figure 6.20: A more difficult reward surface in which only a small proportion of explorations will yield an improvement in reward, once the action unit is on the ridge. By increasing the dimensionality of the space, and reducing the width of the ridge, the learning problem can be made arbitrarily hard for a noise driven hill-climbing process.

Even in the early stages of plasticity, when the algorithm is not hill-climbing but is looking for large regions of above average reward, there is still a problem. As the number of dimensions increases, the size of optimal regions of the output space will become smaller and therefore more likely to be hidden inside larger, less desirable regions.

6.4.9 The Output map as a genetic algorithm

This section on the Output map is now concluded with a brief change of perspective. The basic update rule for the Output map (reviewed on page 174) takes the form:

- If $R > Q(u_1, a_1)$ then update a_1 towards $\langle M_L, M_R \rangle$ proportional to the learning rate, OL.
  i.e.
  $w_1 = w_1 + OL \times (M_L - w_1)$
  $w_2 = w_2 + OL \times (M_R - w_2)$

An alternative rule which results in Output units being attracted to *voluminous* regions of output space above $Mean(r)$ was presented on page 185:

- If $R > Q(u_1, a_1)$ then update a_1 towards $\langle M_L, M_R \rangle$ proportional to the learning rate, OL, and the reward, R.
  i.e.
  $w_1 = w_1 + OL \times R \times (M_L - w_1)$
  $w_2 = w_2 + OL \times R \times (M_R - w_2)$

Finally, it was noted that the criterion for update — If $R > Q(u_1, a_1)$ — could be removed so that the position of a_1 is *always* updated towards $\langle M_L, M_R \rangle$, with the R factor providing the bias towards suitable regions.

Now if this R term is removed altogether from this rule so that the position of a_1 is *always* updated according to:

- $w_1 = w_1 + OL \times (M_L - w_1)$
  $w_2 = w_2 + OL \times (M_R - w_2)$

irrespective of R and whether or not it is greater than $Q(u_1, a_1)$, then we have something of a degenerate case since it appears that there is now no bias in favour of highly rewarded actions. If a multi-unit Output network is now considered, we see that this self-organising map is now fed its own weights as input with a noise signal added. However, a bias towards useful actions is still present by virtue of the preferential selection of certain Output units by the Input units. For any given set of Output units, some will be selected more often because they tend to yield higher Q-values than their neighbours in various situations. Assuming that the updates are made interactively, the Output map will tend to be drawn towards the best actions in the current set. This new set will then be used as the basis for the next round of selection.

The system now bears resemblance to a kind of genetic algorithm (GA) (Holland, 1975) without crossover. In this case the fitness function is effectively the frequency with which a particular Output unit is active. The more often a unit is recruited, the fitter it is, and the more likely the population is to be pulled towards the position of that unit in Output space. Note that this selection pressure is present even with the original update rule, although the 'If $R > Q(u_1, a_1)$' criterion adds an extra bias in favour of highly rewarded actions and removes some of the noise. One problem with removing this criterion is that the Output map tends to drift very easily in the output space effectively on a cushion of noise.

This discussion suggests the possibility of a more sophisticated exploration of the action space involving crossover. For example, new 'proposed' actions could be generated by swapping chunks of weights of successful output units. This would appear to offer the most benefit when the output variables are decoupled. However, it is not clear how this approach would fit in with the self-organisation of the output map, and the idea is not taken further as part of this thesis. The idea of nesting a GA inside an RL system (with the reward signal determining fitness) is precedented in the *classifier systems* of Dorigo and Colombetti (1994).

6.5 Summary

The key issues relating to the convergence, performance, plasticity, stability and efficiency of the architecture are now summarised.

- The standard Q-learning assumptions should be met where possible. In order to maximise the chance of forming an appropriate representation of the input space, the input distribution should change only slowly with respect to the system's free parameters, and converge to a stationary distribution as these parameters are annealed. The reward function ought to be stationary too, although a small amount of non-stationarity may be accommodated by maintaining non-zero learning and exploration rates, at the expense of convergence.

- The contiguity assertion states that nearby regions in input space receive similar rewards for actions close in output space. The greater the extent to which the desired mapping between the input and output spaces satisfies this assertion,

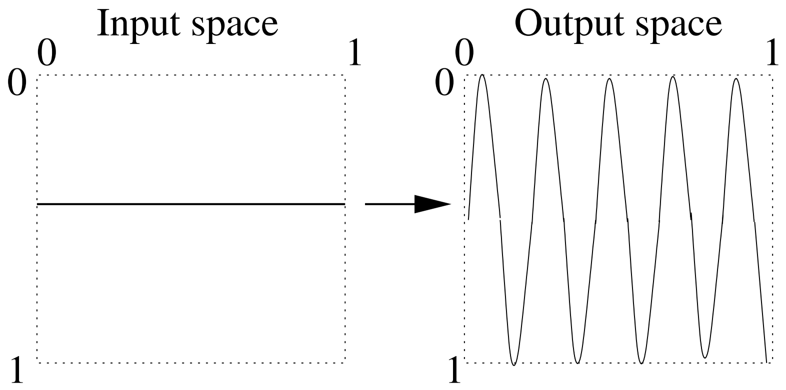


Figure 6.21: A mapping that will be difficult to learn because of the rapid changes in the reward function implied by the oscillations in the output space.

the easier the mapping will be to learn, and the more appropriate the choice of the SOM for the representation of the two spaces. In other words the reward function should vary slowly across the combined input and output space. Consider the mapping of figure 6.21 in which we expect the rapid oscillations in the output space to be hard to learn. Note that this example is different to that of section 5.6.7 because in figure 6.21 the Output space must be mapped *between* the maxima and minima rendering the mapping of figure 5.42(b) ineffective.

- There is no guarantee that positions in the output space that maximise the expected reward will be found by either of the two learning rules of section 6.4.4. Good positions will be more reliably found if they are not hidden within larger regions of lower reward.

- The success of the proposed model depends on an appropriate set of parameters being found. Importantly, the plasticity in each part of the system should be coordinated with respect to the role and needs of the other parts of the system, and the learning task itself. For example, preliminary experiments indicate that while learning in all parts of the system should proceed in parallel, the Input map should approach stability while there is still enough plasticity in the Output map and the Q-learning process to perfect the actions and Q-estimates associated with those stable states.

Appendix D reviews and discusses some suggested heuristics for setting the parameters of the proposed model, including dealing with dynamic environments in the Output map.

With respect to the issue of high reward regions being hidden within large lower regions (as was illustrated in figure 6.10), the covariance learning algorithm proposed by Wedel and Polani (1996) may provide a way to improve performance. By building a local model of the reward surface, the action space can be navigated in a more considered and deterministic fashion, rather than in the highly reactive way proposed thus far. Introducing more modelling is appealing because we have noted that the problem of hidden maxima appears to occur as a result of the stochastic and reactive nature of the search process. As an illustration, consider again figure 6.10, and how performance would change if, prior to unit a_1 being updated, the reward function was sampled until its approximate shape was known. This might lead to the peak being discovered at an early stage of learning, and thereafter being favoured as the optimal point in the action space. This removes some of the reactivity of the algorithm in favour of the more traditional AI components of modelling (the reward surface) and searching (for the optimum, given the model).

When the environment is non-stationary, a similar argument about the relative merits of reactivity vs modelling/planning appears to pertain as for the behaviour based debate. Also, there may be other practical obstacles to the successful application of model and search to the current work. For example, what functional form should the model take? The work of Wedel and Polani (1996) used a model that essentially yielded a direction of largest gradient which then had to be searched. Such an approach is vulnerable to local maxima as well as other difficulties relating to overshooting maxima as a result of getting the search phase wrong. If we adopt a more complex functional form that can model local undulations in the surface then we can expect to require more parameters and therefore also more sample data and more training time. These are important issues because we are seeking an online algorithm which can be run interactively. That is to say that by sampling the environment and taking an action, we expect to move to a new part of the environment. This makes the collection of lots of data about a single point in the action space somewhat problematic, particularly if the environment is changing between visits.

Wedel and Polani (1996) discuss some of these issues, and also identify some other problems with adopting their particular approach. We conclude here that using a modelling approach may ameliorate some of the problems associated with the highly reactive and stochastic nature of the proposed model. However, we also note that the introduction of modelling, search and planning would introduce a new set of issues which would then have to be addressed. These issues are considered outside the scope of this thesis, but certainly justify further investigation.

To conclude the summary, there are clearly other issues regarding convergence that are worthy of study but outwith the capability of the author. For example, it appears to be possible that a unit could fail to climb a continuous slope on the reward surface, even when hill-climbing, because it is easier to slip down the slope than it is to find a potentially very narrow route up it. This need not be a problem for a deterministic hill-climb, but the stochastic nature of the proposed model presents these kinds of difficulties. The root of this problem may stem from the potential situation in which the mean reward of the exploratory region around a_1 is less than the value of the reward *at* a_1. For sufficiently small MA, the reward surface will be a close approximation to a hyper-plane (given C(1) continuity) and hence this situation will not arise. But if the reward surface varies very quickly, then MA may have to be so small to avoid this problem that learning is slowed prohibitively for all practical purposes. This also asks the question of how we detect these situations so that we can select and maintain an appropriate value for MA.

A more complete analysis would be both interesting and valuable, but must constitute Future Work.

6.6 Continuous Actions

The proposed model has been shown to be able to search for and generate real-valued actions from a continuous action space. This is in contrast to coarse coding (instance-based, case-based and nearest neighbour algorithms) where the Q-function is approximated by a number of prototypical Q-values, which then require a gradient ascent search of the action space for an optimal action. However, there is also clearly similarity between coarse-coding, the proposed model, and the SOM model of Touzet (1997), in that the Q-function is represented by a discrete set of sample Q-values. The difference with the proposed model is that a stochastic search of the action space is performed as an integral part of the learning process, which yields a simple lookup problem when it comes to selecting an optimal action for a given state. There are other differences too — particularly in the way that the proposed model maintains an explicit representation of the action space (i.e. one can look inside the representation and see a set of useful actions or symbols), and also the way in which by judiciously selecting the number of action units, the designer is able to capture the structure of the state-action mapping, i.e. one-to-one, many-to-one, one-to-many. For an illustration of the advantages of the latter, see Smith (2001).

However, the similarities lead us to ask whether the representations of the proposed model could be used to generate a true continuous response which varies smoothly with the state information. In other words, can similar kinds of interpolation be employed to generate actions which are not represented directly by any of the units of the action map. A simple approach to interpolation would be to take the current state vector, and estimate an interpolated Q-value for each action based on a weighted sum of the Q-values of neighbourhood state units. This is illustrated in figure 6.22. First, the N nearest neighbours to the current state are identified (using a Euclidean distance measure, for example) . Alternatively, all the neighbours within a fixed radius, d_{max}, may be considered. Next, the interpolated Q-value of taking the action proposed by the first action unit, a_1, is calculated by:

$$Q_{interp}(query - state, a_1) = \sum_{i=1}^{N} \frac{G(d_i)}{\sum_{j=1}^{N} G(d_j)} \times Q(s_i, a_1) \tag{6.6}$$

where G could be any smooth kernel function that weights smaller distances with a heavier value. Interpolated Q-values are then calculated for every possible action, and the one with the highest interpolated value is chosen. This is only slightly more sophisticated than simply taking the action proposed by the nearest state unit, but does allow interpolation to be performed in the state space at little extra computational cost.

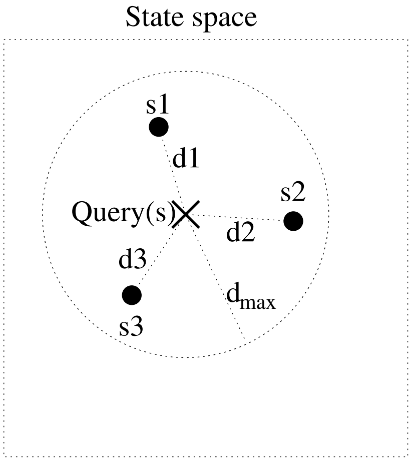


Figure 6.22: Interpolation between stored Q-values. The current state is shown as the query point, for which three nearest neighbour state units have been identified — s_1, s_2, and s_3. Interpolation can now be performed over the state space by suitably combining the Q-values attached to these three states according to the distances — d_1, d_2 and d_3 — from the query state.

However, what we are really interested in is interpolating in the *action* space, since then a smooth input-output mapping may be achieved. The simplest approach would be to exploit the topology of the Output map in the following way: For any current state vector, first identify the winning state unit in the usual way (the one with the smallest Euclidean distance). Next, line up all possible actions in the order they appear in the Action map, as in figure 6.23. Finally, choose some parameterised functional form, approximate a continuous partial Q-function from the winning state, and select the action at the global maximum of this function. This approach requires that a suitable functional form is known, and that topology is indeed preserved in the action map. Another assumption is that equally spacing the action units along the x-axis is sensible. An alternative would be to space them according to their Euclidean distance in action space. An advantage of using this approach is that only a one-dimensional function need be considered (assuming a one dimensional Action map).

More thorough interpolation could be achieved by searching the action dimensions of the combined state-action space for high Q-values in a manner similar to the instance-based and nearest-neighbour techniques discussed back in chapter 4. Given that we can interrogate the Q-function at *any* state-action index (even those not explicitly represented) using a kernel weighting function (such as that of equation (6.7) for example),

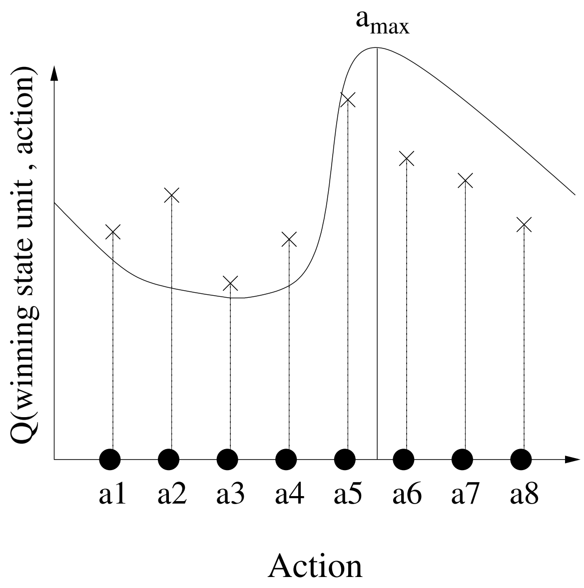


Figure 6.23: For the winning state, a parameterised functional form is fitted to the sample Q-values associated with the current action set. Here the optimum action is estimated as a_{max}.

then we can effectively perform a hill-climb of the action space to discover optimal actions.

$$Q_{interp}(s_{query}, a_{query}) = \sum_{i=1}^{N} \frac{G(d_i)}{\sum_{j=1}^{N} G(d_j)} \times Q(s_i, a_i) \tag{6.7}$$

where $Q(s_i, a_i)$ with $i \in 1 \dots N$ are the Q-values of the N-nearest explicitly stored state-action pairs to the query state-action pair. As before, d_i is the Euclidean distance from the i^{th} nearest neighbour to the query point (in the combined state-action space this time), and G is some smooth basis function which gives a heavier weight to smaller distances.

The idea is illustrated in figure 6.24. However, this approach is computationally expensive, particularly in high dimensions. This is significant because the search must be performed every time an action is sought. We also expect a certain degree of vulnerability to local minima (as suggested in the figure). Notwithstanding these criticisms,

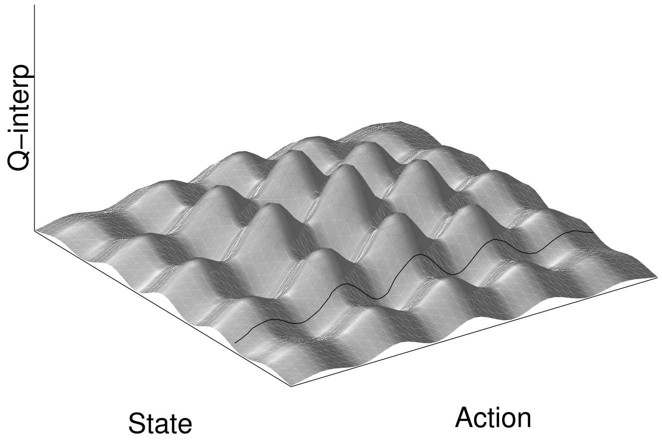


Figure 6.24: Given a function Q_{interp}, which can be used to interrogate the Q-function at any real-valued state-action index (by interpolating between known Q-values using a kernel function similar to equation (6.7)), then for any given state, a hill-climbing algorithm may be performed in the action dimension(s) to discover the optimal action for that state. The solid line represents the search to be performed in this example in which both the state and action spaces are one-dimensional.

this does represent one possible approach to producing a continuous response through interpolation.

If the above algorithm was implemented, how would we expect the model to perform compared with the coarse coding algorithms? After all, now the two approaches look very similar with both involving interpolation and search over stored prototypical Q-values. However, in the proposed model there is still the primary search of the action space being performed as an integral part of the normal learning process. This search is costly (in that we have to perturb desirable actions in order to explore the space), but one advantage is that the prototypical Q-values stored will be particularly relevant to solving the task. Hence we could view the interpolating version of the proposed model as a special case of coarse-coding where the Q-function is represented at a higher resolution in potentially profitable regions of the space, with the search for 'profitability' taking place during learning. The idea of biasing the representation of the Q-function is a central theme of many approaches, including the experiments of Santamaria et al. (1997) in which a (static) variable resolution approach to coarse coding is investigated.

6.7 Comparison

We now conclude this chapter with a look at the achievements of the new model. Chapter 4 reviewed a number of existing approaches to representing and generalising over the action space of RL problems. A set of desirable features were then distilled out of that discussion and presented at the beginning of chapter 5. Recall that the discussion recommended that generalisation be dynamic and adaptable, that the standard RL theory be closely adhered to maintaining a central concept of estimating expected reward, that real-valued states, actions and rewards be supported, and that actions discovered for one part of the input space should be reusable in other parts. Having a choice of different actions in each state was also deemed important, and the natural ideals of simplicity, efficiency and scalability were considered too. Justifications for aspiring to these properties were offered as part of the review of the alternative algorithms back in chapter 4.

Figure 6.25 offers an explicit comparison of existing approaches with respect to the desired criteria. The contents of the table essentially represent a summary of the discussion of chapter 4, with the final column suggesting the progress made in chapter 5. Producing such a table is a challenge since there are many grey areas, continuous sliding scales, and underspecified descriptions that make some elements of the table hard to assess. Additionally, different algorithms are usually intended to address tasks with slightly different boundaries so an equitable comparison is problematic. Comparison is often made more difficult by algorithms that are so different that it is not clear that certain criteria are so relevant. For example, the criterion relating to ease of interpolation is clearly far more germane to those techniques which rely on discrete prototypes (such as coarse-coding and the SOM-based methods) than to the backpropagation methods in which interpolation is achieved largely gratis.

It is therefore noted that although question marks are used to denote uncertainty, no doubt almost every element of the table could potentially be debated. Indeed in the absence of any rigorous theoretical or empirical comparison, assessment of individual models may be largely subjective. For example, in a similarly motivated table in Moore (1990), CMAC is judged to have efficient memory usage despite the fact that the number of tiles increases exponentially with the dimensionality of the problem.

		Coarse-coding					SOM			Backpropagation						
		Handcoding (Mah. & Con.)	Coarse-coding	RBF (Santamaria)	CMAC (Prescott)	Nearest Neighbour (Moore)	SOM (Touzet)	Motoric map (Ritter et al.)	Covariance learning (Wed. & Pol.)	CRBP (Ackley et al.)	CRBP (Ziemke)	QCON (Lin)	Backprop (Touzet)	SRV units (Gullapalli)	Q-AHC (Rummery)	PROPOSED MODEL
	Dynamic generalisation	✗	?	?	✓	?	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓
Generality	Adhere to RL theory	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✓	✗	✓	✓	✓
Generality	Real-valued, discounted reward	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✓	✗	✓	✓	✓
Generality	Real-valued states and actions	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
Flexibility	Re-use of actions	✓	?	?	✗	?	?	✗	✗	✗	✗	✓	✗	✗	✗	✓
Flexibility	Multiple actions for each state	✓	✓	✓	✗	✓	✓	✗	✗	✗	✗	✓	✗	✗	✓	✓
Flexibility	Interpolation	✓	✓	✓	✗	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗	✓
Scalability	Low update cost	✓	?	?	?	✓	✓	✓	?	✓	✓	✓	✓	✓	✓	✓
Scalability	Low access cost	✓	✗	✗	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
Scalability	Low memory use	?	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✗
Scalability	Scalable	?	?	?	?	?	✗	?	?	✗	✗	?	✗	?	?	?

Figure 6.25: Some existing approaches to action space generalisation are compared with the proposed model with respect to the identified set of desirable properties. The main part of the table is duplicated from figure 5.1. The algorithms are grouped into non-neural, SOM-based, and backpropagation-based for convenience. A tick indicates that the algorithm satisfies the criterion (or performs favourably in comparison to the other models), a cross that it does not, while a question mark represents uncertainty. This table should be interpreted as a guide for further discussion, rather than an authoritative last word on the success of failure of specific algorithms (see text).

A detailed debate of every element of this table would simply be beyond the scope of this document. However, a brief discussion of some of the more relevant findings is offered. Firstly, static representations such as those used in Mahadevan and Connell (1991) are considered to be outside the scope of this thesis which is explicitly concerned with *online generalisation*. Similarly, model-based learning is not considered which is why Tesauro's *TD-Gammon* does not feature in the table.

The simplest non-neural approaches score well, but lack adaptability and scalability. The existing SOM-based approaches exhibit inflexibility, and most of the backpropagation approaches are forced to make significant departures from the underlying theory in order to accommodate an ostensibly supervised technique within an RL framework[8]. Notable exceptions are the work of Gullapalli (1990) and Rummery (1995) which, along with Lin's QCON algorithm[9] are considered good foundations upon which to build future work. To this end, a more comprehensive comparison of the proposed model with Gullapalli's model is undertaken in chapter 8, and the whole issue of the use of backpropagation is also tackled in more detail there. In the meantime, flexibility in both action exploration and action selection emerges as a consistent weakness of these approaches[10], and there is also doubt over their scalability. This last point pertains to potential problems with local minima, long training times, and other issues relating to interpretation, diagnosis and robustness to non-stationary environments. These are not necessarily catastrophic flaws, and are considered in much more detail as part of chapter 8. However, in compensation for the negative features of backpropagation, the efficient memory use of all these approaches is a clear advantage. In general no algorithm is considered to have proved itself truly scalable in a broad range of circumstances. In the later section on future work we will consider how some of these approaches may be combined in order to achieve the best of different worlds.

One element of the table that is particularly worth explaining is the update cost of Wedel and Polani's Covariance Learning algorithm. In fact, every model is considered

[8]The CRBP algorithms and Touzet's backpropagation approach are considered unscalable because of their assumptions regarding the appropriateness of a binary reward signal, binary outputs, or the idea of learning towards complements of punished actions as a means of driving learning.

[9]Dynamic generalisation needs to be addressed here.

[10]Lin's QCON model is a clear exception, and will be considered again as part of the section on future work (section 9.2.6).

to perform adequately with respect to this criterion (some performing better than others), but the reliance of Covariance Learning on collecting multiple samples for each state before a model of the reward surface over the action space can be constructed is an added complication that may or may not yield adequate compensation in terms of performance.

The proposed model achieves most of the objective criteria, although there are again concerns over scalability which will be considered in later chapters. In particular, the local representation of the SOM, the requirement for data with low intrinsic dimensionality, and the fact that each input line contributes equally to the winner selection metric are all potential problems. Unfortunately though, simply adding up the pros and cons in each column will not be a decisive evaluation strategy. We note that different rows carry different weight in different circumstances, that there is significant room for interpretation in each element of the table, and that some algorithms may easily be adapted to embrace more of the criteria. However, this table does succeed in highlighting the current position and direction of the thesis with respect to existing techniques.

CHAPTER 7

Learning Ballistic Problems

7.1 Introduction

The proposed model has been shown to make provision for learning real-valued actions from a continuous output space. A simple application is now presented which allows the properties of the algorithm to be explored in more detail. This chapter will also provide a basis for a comparison between SOM and backpropagation based models in chapter 8. The experiments of this chapter are concerned primarily with the qualitative performance of the Output map, and less so with the quantitative performance of the system as a whole. In particular, reward graphs will not be shown since they will add little to the experiments and discussions of chapter 5.

7.2 A new task

In the introductory chapter it was noted that some learning tasks will benefit from being able to adapt to real-valued actions, and some others will actually *require* adaptive

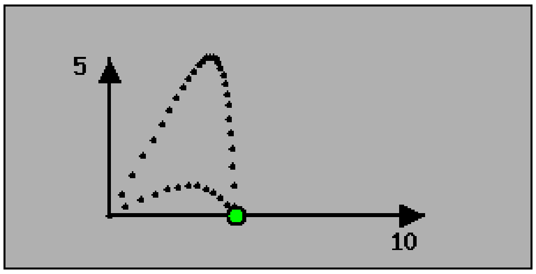


Figure 7.1: Two sample trajectories out to a target positioned four units along the horizontal axis.

real-valued actions. The application of this chapter falls into the second category, since there will be no opportunity to approximate real-valued actions by interleaving discrete ones.

The modelled environment is based on the problem of throwing projectiles. Consider a javelin thrower standing at the origin of a coordinate system, and a target lying somewhere along the positive x-axis. The javelin thrower must select an appropriate horizontal and vertical throwing speed (U_x and U_y respectively) at which to throw the projectile so that it lands as close to the target as possible. Figure 7.1 shows an example of two different trajectories to a target positioned four units along the horizontal axis. The initial horizontal and vertical velocity components are restricted to the range $[0,1]$ which limits how far the projectile can be thrown. The labelling on the axes reflects this. After each throw, a reward is given that is equal to the negative of the distance between the landing point of the projectile and the target. Hence a direct hit results in a reward of 0, while a wild miss could result in a reward of up to -10.

The environment model is based on the mechanics of the physical world with the following parameters. Note that the mechanics of the problem are arbitrary and the physically inspired model just happens to be convenient for generating an interesting, parameterised problem.

Parameter	Value
Mass of projectile, M	1
Gravity, G	0.05
Coefficient of friction due to air resistance, Fr	0.1
Horizontal wind speed against the thrower, W	0
Horizontal position of target, X	4

Table 7.1: The parameters of the system along with some initial values. Units of measurement are implicit and not specifically intended to follow any particular real physical system. Clearly terms like *air resistance* and *gravity* enjoy only metaphoric sense here.

By the laws of mechanics, the vertical position, s_y, of the projectile at time T is given by:

$$s_y(T) = \int_0^T U_y - t \times G \ \ dt \quad = \quad U_y T - \frac{1}{2} G T^2 \tag{7.1}$$

Note that air resistance is not modelled in the vertical component. However, it is modelled in the horizontal component. The horizontal force on the projectile due to air resistance, f_x, is assumed to be proportional to the horizontal speed of the air relative to that of the projectile. Hence:

$$f_x(t) = Fr \times (v_x(t) + W) \text{ , where } v_x(t) \text{ is the horizontal velocity at time } t.$$

Since the mass of the projectile is 1 for convenience, then the horizontal acceleration at time t is give by:

$$a_x(t) = -f_x(t) \qquad \text{and so} \qquad \dot{v}_x(t) = -Fr(v_x(t) + W)$$

This is a linear differential equation with solution:

$$v_x(t) = (U_x + W)e^{-Frt} - W$$

Now this can be integrated to give the horizontal distance travelled by the projectile at a particular time T:

$$\begin{aligned} s_x(T) &= \int_0^T (U_x + W)e^{-Frt} - W \quad dt \\ &= \frac{(U_x + W)(e^{-FrT} - 1)}{-Fr} - WT \end{aligned} \tag{7.2}$$

Now the horizontal landing position can be calculated:

By equation (7.1), $T_{landing} = \frac{2U_y}{G}$, and then substituting this into equation (7.2) gives:

$$s_x(T_{landing}) = \frac{(U_x + W)(e^{-Fr(\frac{2U_y}{G})} - 1)}{-Fr} - W(\frac{2U_y}{G}) \tag{7.3}$$

Then:

$$\text{reward, } r = -\left|X - s_x(T_{landing})\right| \tag{7.4}$$

The goal of the system in the following experiments will be to learn to produce appropriate horizontal and vertical throwing components that hit the target under various conditions. Note that the learning system will not have direct access to the mechanics given above. The only input to the system will be zero or more of the environment parameters from table 7.1. The only feedback the system will receive will be the reward of equation (7.4). The model details are included here for reproducibility rather than specific interest, since the choice of mechanics is somewhat arbitrary. The main point is that we have created a parameterised problem requiring real-valued actions to use in this chapter and chapter 8.

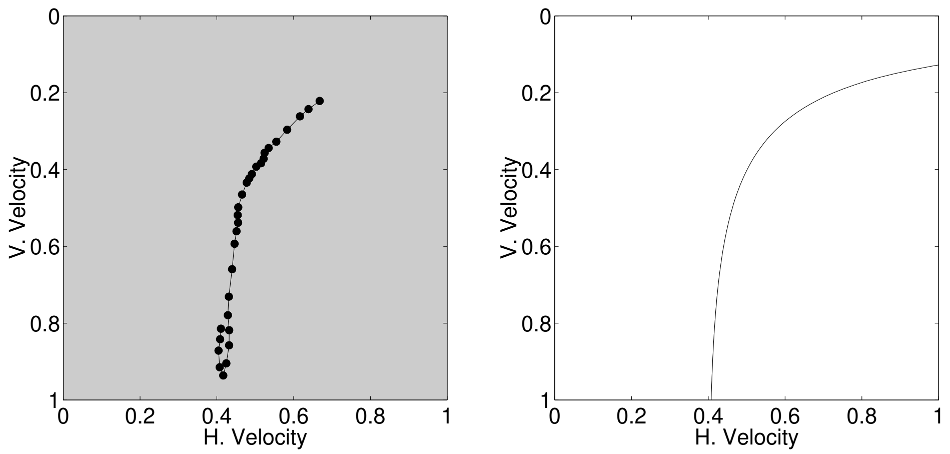
7.3 Experiment I

Assuming the parameters of table 7.1 are constant, then there is effectively only one state to the problem, and since only a single trajectory is required to solve the problem for a given target position, X, then only one Output unit is required too. Later we will allow the parameters of table 7.1 to vary thus restoring the concept of a state space. Reward is immediate so the reward horizon can be set to one, and the only other learning parameters of this degenerate case are given by:

Parameter	Value
Q-learning rate, α	$f(t)$
Learning rate of Output map, *OL*	$f(t)$
Max. Exploration distance around Output unit, *MA*	$f(t)$

with $f(t)$ annealed from one to zero in the usual way depending on how much learning is required. For this experiment $f(t) = 0.99^{throw}$ was found to be close to optimal. As usual, 'optimal' is defined as the fastest annealing schedule possible that does not compromise final performance (in terms of the final average reward received). This annealing rate must be selected by hand. When the system is trained on this degenerate case, the single action unit learns an appropriate throwing velocity to hit the target with an average error of 0.02 of a unit (along the x-axis) within a few hundred training throws. The task is easily solved since the single Output unit only has to perform a hill-climb (no local maxima) on the reward surface over the two dimensional action space in a manner following the simple system analysed in chapter 6.

However, this basic experiment is made more interesting if more units are added to the Output map. A second experiment is performed with the following additional parameters:



(a) Output map after learning.

(b) Analytical solution.

Figure 7.2: Numerical and analytical solutions to the problem of maximising the reward of equation (7.4) under the parameters of table 7.1. Figure (a) shows the Output map after learning. Figure (b) shows the analytical solution to the same problem.

Parameter	**Value**
Output map size	30×1
Output map neighbourhood size, *ON*	$15 \times f(t)$
Probability of Q-Exploration, p	$f(t)$

Now, $f(t) = 0.999^{throw}$ is used because there are more Q-values and Output unit positions to learn. In addition, minimum values are maintained for some parameters to achieve the best looking maps. These are 0.1, 0.05, 0.5, 0.1 and 2 for *OL*, α, p, *MA*, and *ON* respectively. The numbers are somewhat arbitrary, although the unusually high minimum value of p ensures that all Output units are involved in solving the problem. This allows us to highlight the potential for learning multiple solutions.

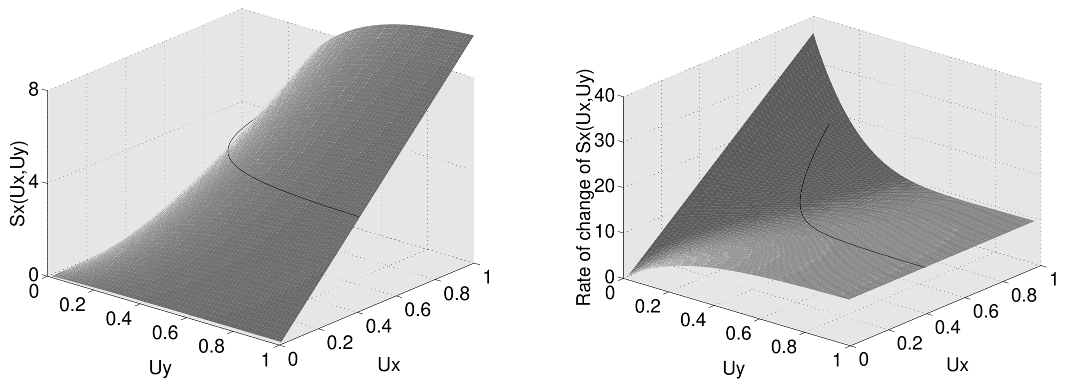
Figure 7.2(a) shows the Output map plotted in the two dimensional output space after parameter annealing (approximately 4000 throws). The units have identified thirty different trajectories for reaching the target ranging from low, hard throws to high lobs. Now, by rearranging equation (7.3), the relationship between U_x and U_y can be derived for which the target is hit by the projectile:

$$U_x = -Fr\frac{X + \frac{2WU_y}{G}}{e^{-Fr(\frac{2U_y}{G})} - 1} - W \tag{7.5}$$

Figure 7.2(b) shows this relationship plotted for all real values of U_x and U_y (where both lie in the range $[0,1]$), which neatly corresponds to the set of solutions found by the learning system itself. Interestingly, although the RL system clearly adheres to the analytical solution, it appears to be reluctant to use the very hardest and lowest throws which correspond to the missing part of the curve in figure 7.2(a). The reason for this probably lies in the variable reliability of different types of throw. To illustrate this, consider first figure 7.3(a), which shows the landing position of the projectile as a function of throwing velocity (same model parameters as in table 7.1), and then figure 7.3(b), which shows the first derivative of this function. The observation is that for the very lowest and hardest throws, the rate of change in the landing position with respect to the throwing velocity is largest. This means that small deviations in the throwing velocity as a result of the random noise generated by exploration, will have the greatest impact at this point. In other words, given that there is noise in the system, low hard throws are the least reliable way to hit a target. The Q-values will reflect reliability because they are a measure of *expected* reward. Given this explanation, the behaviour of the system is considered desirable.

Figure 7.4 reports the actual and analytical solutions of a second learning trial in which the wind speed and gravity are increased (Wind = 0.3, G = 0.1). A satisfactory correspondence is again observed, with a similar aversion to the lowest and hardest throws for reaching the target.

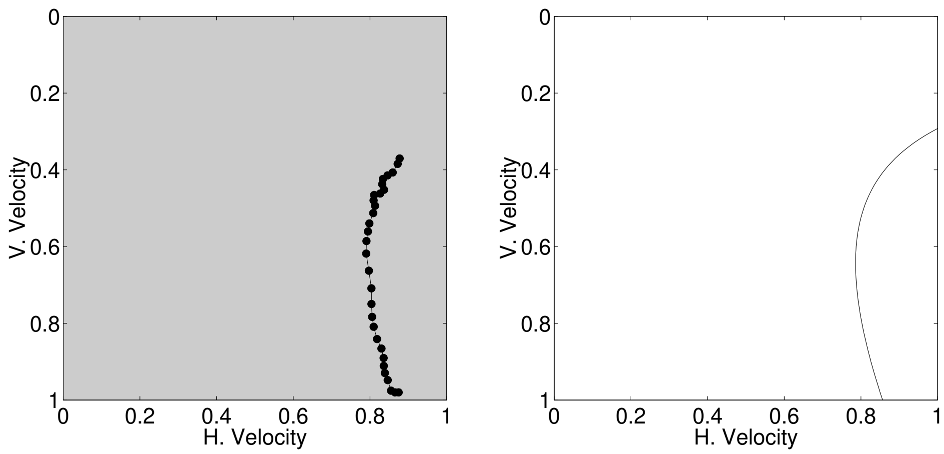
Note that generally there is no reason why the *entire* range of actions should be discovered since any subset of points, or indeed any single point on each curve will be sufficient for solving the problem. The graphs shown illustrate the potential for discovering multiple actions given the stochastic nature of the search process, but if *all* possible solutions are required with certainty then extra care must be taken to ensure adequate exploration of the space (achieved here through a large value of the exploration parameter, p).



(a) Landing position of the projectile as a function of the throwing velocity. The solid line denotes all trajectories that reach a target at $X = 4$.

(b) Numerical approximation of the first derivative of the landing position with respect to the throwing velocity. Again the solid line denotes all trajectories reaching a target at $X = 4$.

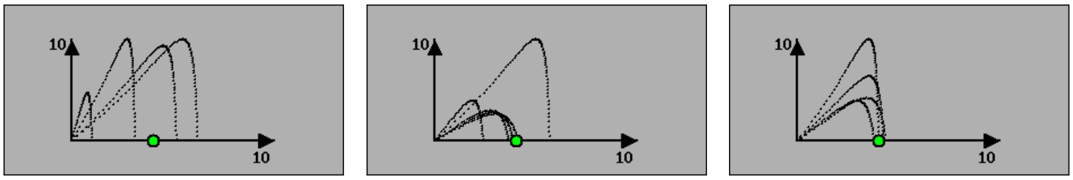
Figure 7.3: Graphs show how the landing position and the gradient of the landing position vary with the throwing velocity.



(a) Output map after learning.

(b) Analytical solution.

Figure 7.4: Learned (a) and analytical (b) solutions to the problem of maximising reward when $G = 0.1$ and $W = 0.3$ (other parameters as defined in table 7.1). This time more throwing velocity is required, but the correspondence between the analytical and numerical solutions is still apparent.



(a) Before learning.

(b) During learning. Some hits, some errors due to exploration.

(c) After learning. Exploration is minimal. Different solutions to the same problem are found.

Figure 7.5: Throwing the projectile as learning progresses.

For the purposes of qualitative illustration, figure 7.5 shows how the agent learns to throw the projectile with increasing accuracy as the trial progresses.

7.4 Experiment II

The problem is now extended by parameterising the task with respect to the position of the target along the x-axis. This changes the problem to one involving a single-dimensional state space. To map this space, a one-dimensional Input map of 30 units is used, with an initial neighbourhood of 15, annealed in the usual way. In this experiment minimum values are no longer used for any of the parameters. The new (or altered) parameters are given as:

Parameter	Value
Learning rate of Input map, IL	$0.3 \times f(throw)$
Input map neighbourhood size, IN	$15 \times f(throw)$
Annealing schedule, $f(throw)$	0.9985^{throw}

Three runs of this new experiment are shown in figure 7.6. In all cases, the Input map accurately and smoothly represents the range of possible target positions by equally spacing the thirty states in the single dimensional state-space. the Output maps vary across the runs, but recall that each target position is satisfied by a range of throws within the space. This gives the Output map a large amount of freedom to represent this space, and in fact all runs performed well (figure 7.7, to come).

7.4.1 Topology preservation

We have already seen the advantage of neighbourhood Q-learning in chapter 5, but this experiment gives us another opportunity to quantify the benefit.

Figure 7.7 shows three learning curves. The first to consider is the one labelled $+NL/0.9985$ which corresponds to the previous experiment, utilising neighbourhood Q-learning and an annealing schedule of 0.9985^{throw}. This was empirically found to be the fastest annealing rate that did not compromise final performance. In other words the base level of this curve can be considered as the model's optimum performance. The same experiment without neighbourhood Q-learning yields the curve labelled $-NL/0.9985$, showing a marked decrease in both learning speed and final performance. It turns out that learning is so handicapped by not using neighbourhood Q-learning that the annealing schedule must be decreased to 0.9997^{throw} before final performance comes close to achieving that of the original experiment (see curve $-NL/0.9997$). The annealing schedule numbers of 0.9997 and 0.9985 are rather meaningless on their own, so the actual schedules are superimposed on the graph. The main result is that neighbourhood Q-learning is able to expedite convergence to optimal performance by a factor of at least six. Interestingly, this speedup factor appears to be consistent with the approximation obtained back in figure 5.14, even though that experiment involved a very different task.

7.4.2 The cost of organisation

Looking again at the Output maps of figure 7.6, one aspect that does vary is the degree of clustering. For example, in the second run the Output units are significantly more evenly distributed than in the third. This can be explained by the observation that there are essentially two forces present in the Output map. The first is the attraction of units towards highly rewarded regions of the space, and the second is the pull that

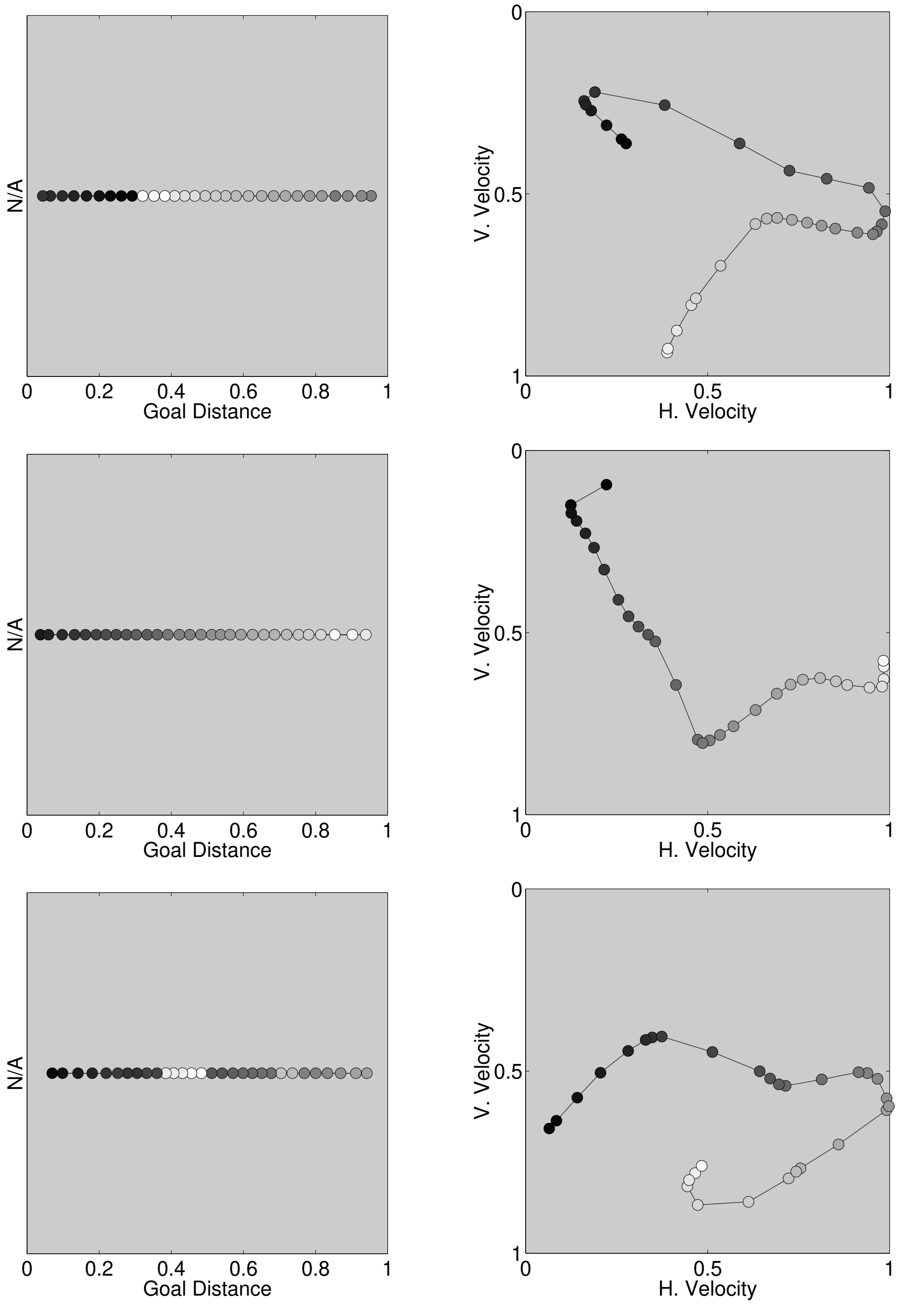


Figure 7.6: Results of three runs of the same experiment in which the system attempts to maximise the reward of equation (7.4) under the parameters of table 7.1, but where the distance of the target is varied (drawn randomly and evenly from the range $[0,1]$) and supplied as an input to the system before each throw. The learned Input map of each run is shown on the left (shown in two dimensions for convenience), and the Output map on the right. Output units are coloured simply according to their index within the map. Input units are colour coded according to the Output unit for which they have the highest Q-value.

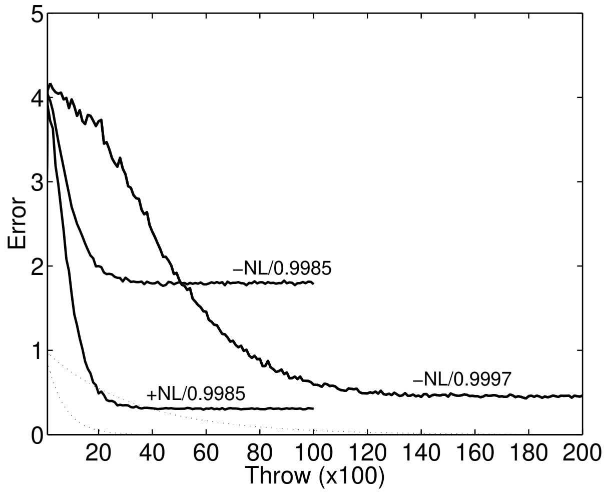


Figure 7.7: Three learning curves are shown plotting average Error (the negative of average reward) over the number of throws. Each curve is labelled according to whether neighbourhood Q-learning is used (+NL) or not (-NL), and according to the annealing schedule used. The two different annealing schedules (0.9997^{throw} and 0.9985^{throw}) are also plotted as the dotted lines, with the steeper schedule naturally corresponding to 0.9997^{throw}. Each curve is averaged over a large number of runs, and so the error bars can be considered negligible for comparison purposes.

neighbours exert on each other as imposed by the familiar SOM learning rule[1]. This second force is necessary because it ensures competition between regions of the output space for Output units, thereby encouraging *all* units to occupy popular and therefore presumably useful regions. This second force also encourages topology preservation which, as we have seen, can be used to expedite the passing of reward information around the Q-table through neighbourhood Q-learning. However, the force exerted on an Output unit by its neighbours is also disruptive since it will tend to drag that unit away from its preferred position — a position that it is attempting to discover for itself through a costly trial and error process. The instances where clusters of units have appeared along the Output map in figure 7.6 are probably the result of this topological force overpowering the exploratory force at that point. It is important to strive for a balance between these two forces that respects both the individual unit and the society of the map as a whole. This will then allow a suitable tradeoff between competition/organisation in the output space, and the ability of units to independently explore this space.

[1]Note that this second force is quite distinct from *neighbourhood Q-learning* which is only concerned with the Q-table, not the position of Output units.

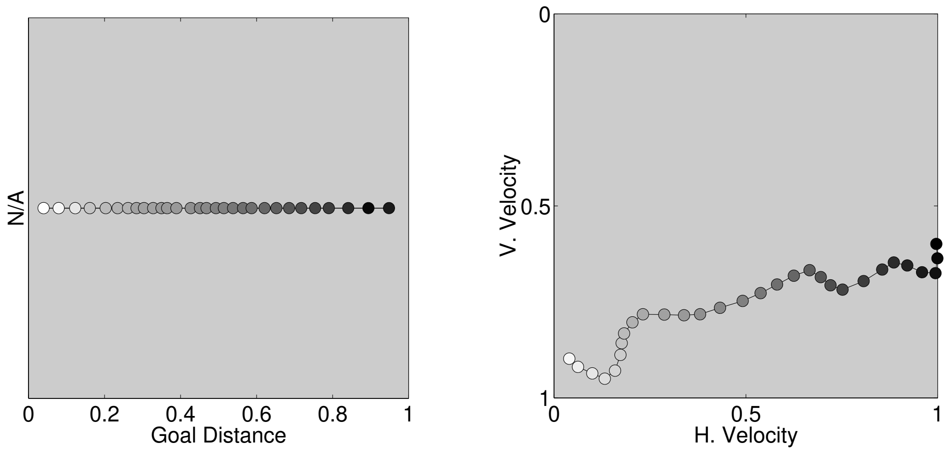


Figure 7.8: Input and Output maps for the same setup as figure 7.6, except that a *larger* neighbourhood of $25 \times f(throw)$ is used.

As an example consider figures 7.8 and 7.9. The graphs show the Input and Output maps for similar experiments to those reported in figure 7.6, except that the size of the neighbourhood is varied. Figure 7.8 shows the effect of increasing the neighbourhood and therefore the 'second force' referred to above. The consequence is a more continuous Output map with greater topology preservation. Figure 7.9 shows the effect of using a smaller neighbourhood. Here the Output map is less well organised, topology is compromised in the Output map, and there is greater discontinuity in the mapping between the Input and Output maps.

The SOM has a general tendency to minimise its final length (defined as the sum of the distances between nearest neighbours, in a two dimensional map), and indeed this is a feature that has been exploited to provide approximate solutions to the Travelling Salesman Problem for example (see Favata and Walker (1991)). In this sense, the Output map of figure 7.8 has sought the most consistent set of throws for reaching a variable target. Although the Output map of figure 7.8 has a more intuitive appeal than that of figure 7.9, we note that if the neighbourhood force becomes too strong, then the disruption to units by their neighbours will actually outweigh the constructive capabilities of the map, and the map will tend to collapse into highly active regions of the Output space. As a final point, we note that the actual performances (in terms of the final reward average) associated with figures 7.6, 7.8 and 7.9 are very similar, and

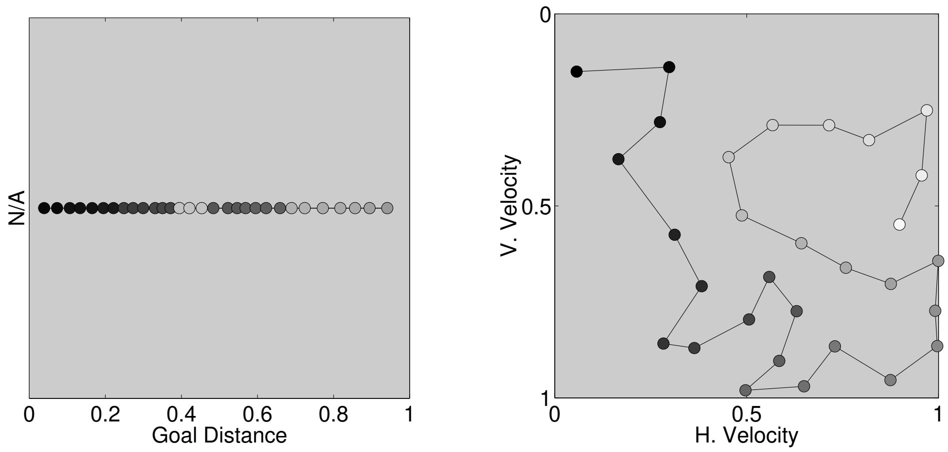


Figure 7.9: Input and Output maps for the same setup as figure 7.6, except that a *smaller* neighbourhood of $5 \times f(throw)$ is used.

we should not be unduly prejudiced by the appearance of figure 7.9.

7.5 Summary

This chapter has sought to explore some qualitative performance issues pertaining to the Output map in more detail. The projectile throwing environment, which will be used in the next chapter to compare the proposed model with an alternative based upon backpropagation, was seen to necessitate the adaptation of real-valued actions since in this instance it is not possible to approximate a real-valued action with a series of discrete actions. Further more, fixing a set of actions in the output space at a resolution that would have achieved comparable performance to the proposed model, would have led to hundreds of irrelevant units, thus crippling the exploratory process. In higher dimensional spaces, handcoding actions to cover the entire space is clearly even less viable.

First we saw the model's potential for generating and maintaining *multiple* actions for the same state. This feature is deemed to have positive implications for robustness, particularly if we expect situations in which some actions become unavailable for reasons perhaps outside the control of the system. Examples might be as a result of damage, or on the advice of another module in a behaviour based system.

Next we reiterated the advantage of using topology preservation in both the Input and Output maps to expedite learning, by comparing the proposed model with one in which neighbourhood Q-learning was not used. Interestingly, the speed up factor of approximately six was found to be consistent with the same comparison performed on a different experiment in chapter 5. In general, we might expect the smoothness of the reward function to control the degree to which neighbourhood Q-learning is advantageous.

Finally, we briefly investigated the effect of varying the neighbourhood size on the continuity or *consistency* of the Output map. As we might have expected, increasing the neighbourhood size leads to more consistent maps in which the length of the map is minimised. While we suggest that consistency in the Output map is beneficial for neighbourhood Q-learning, and also, potentially, for interpolating between actions, there was no immediate evidence for Output map consistency leading to an improvement in final performance. Moreover, we should be aware that if too much neighbourhood learning (different from neighbourhood Q-learning, which is exclusively concerned with the Q-table) is used so that it overpowers the exploratory capabilities of individual units, then the map may collapse onto highly active regions of the output space, with disregard for their infrequently active neighbouring regions. Hence in this respect a balance must be struck.

CHAPTER 8

Backpropagation

8.1 Introduction

At the end of chapter 6 the proposed model was briefly compared with a number of alternative approaches to continuous action-space generalisation. Out of all the algorithms reviewed, the QCON algorithm of Lin (1993) (section 4.6), the SRV unit of Gullapalli (1990) (section 4.9), and the Q-AHC model of Rummery (1995) are considered the most promising. All maintain estimates of expected reward, support real-valued inputs and outputs, and appear to offer good scalability. All three are also based on backpropagation suggesting that this particular approach is worth investigating further. Although interesting, Lin's model is not directly relevant to the current problem because a set of actions must be fixed beforehand, so the emphasis will be on Gullapalli's SRV unit, and the related Q-AHC approach. Although other MLP-based approaches such as CRBP (Ackley and Littman (1990); Ziemke (1996)) and Touzet's algorithm (Touzet, 1997) may also be worthy of comparison, they are rejected here on the grounds that they place too much emphasis on binary reward signals and the use of complementary actions for driving learning.

Tesauro's *TD-Gammon* has already proved that backpropagation can be used to good effect in RL problems, although this particular approach is not directly applicable here because it requires an environment model. Further justification for investigating backpropagation within the current context is that it is ostensibly a supervised learning technique and as such represents something of an interesting paradigmatic alternative to the SOM. As we will see, there are also some other key differences between backpropagation and the SOM, apart from the degree of supervision, that will impact heavily on their suitability for various RL problems.

8.2 Adapting backpropagation for RL

Backpropagation has an elegance and power that has been exploited over the last fifteen years in a wide range of applications as diverse as handwritten digit recognition (LeCun et al., 1990), learning to play backgammon (Tesauro, 1994), learning to pronounce English words from an orthographic description (Sejnowski and Rosenberg, 1986), scheduling problems (Crites and Barto, 1996), modelling brain function (Shillcock and Monaghan, 2001), and robot control (Lin, 1991), to name only a few. But out of this body of research some now well known problems have emerged including local minima, selection of appropriate initial weights and learning rates, the challenge of interpretation and diagnosis, and perhaps most notably, long training times (Rojas, 1996; Bishop, 1995). However, before the benefits and drawbacks can be compared within the current context of reinforcement learning of real-valued functions, the basic algorithm first needs to be augmented to remove the assumption that appropriate input-output pairs are explicitly available — an assumption that does not hold in the standard RL formulation.

The CRBP algorithm (Ackley and Littman, 1990) circumvents this problem by introducing a binary reward signal with learning taking place in the direction of the complement of the binary output vector in the event that this reward is negative. However, it has already been suggested that binary reward and outputs are potentially restrictive. A simple alternative model is proposed in figure 8.1. The system consists of two networks. The network on the left computes an action for each input, while the network on the right simultaneously computes the estimated expected reward of taking that action under that same input.

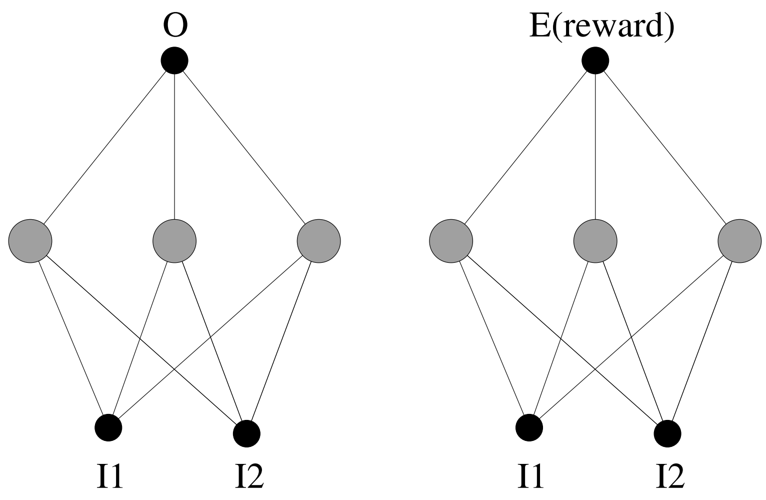


Figure 8.1: Two backpropagation networks compute a reinforcement learned mapping from two inputs to a single output. The network on the left — the *actor* network — computes the mapping from inputs to output while the network on the right — the *critic* network — maintains a partial Q-function which maps an input vector to the estimated expected reward of taking the action proposed by the *actor* network.

This model is proposed for the purposes of this chapter, and is to enable a preliminary comparison between backpropagation and the SOM for generalisation in reinforcement learning problems. We are not attempting to introduce a new architecture here — indeed the model of figure 8.1 has many precedents in the literature. One such precedent is clearly Gullapalli's SRV unit which embraces the same idea but uses only a single linear unit in the place of each network. This simplicity of SRV appears to be a handicap since the original model has trouble with relatively straightforward mappings (Gullapalli, 1990). However, we consider the concept to be basically sound as well as relevant for a comparison with the proposed model of this thesis. We intend to present the model of figure 8.1 as a more powerful alternative to SRV which can act as a suitable representative for backpropagation in the current context of comparison with the SOM-based approach considered thus far.

Figure 8.1 has other precedents in the literature too. One is the *Associative Reinforcement Comparison* (ARC) algorithm of Sutton (1984) (reviewed in Kaelbling et al. (1996)) in which the same idea is used, except that binary actions are maintained in a manner not dissimilar to CRBP. Indeed the ARC, along with other connectionist approaches to RL generalisation, appears to have inspired Gullapalli's SRV algorithm, with the latter's contribution being the provision of real-valued actions. Another relevant precedent is the Actor-Critic model of reinforcement learning which was an early model first proposed in Barto et al. (1983) (reviewed in Sutton and Barto (1998)), and later adapted to real-valued actions in Rummery (1995) (as described in section 4.10).

To clarify matters for the purposes of this and the following chapter, and without labouring the possibilities of which model inspired which, let us henceforth refer to the scheme of figure 8.1 as the *actor-critic* model, noting that its primary inspiration came from SRV, although a retrospective look at the literature reveals a number of important precedents including ARC of Sutton (1984), the AHC model of Barto et al. (1983), and the extension to real-valued actions in the Q-AHC model of Rummery (1995). We reiterate the purpose of this chapter not as the introduction of a new model, but as a comparison of the computational features offered by backpropagation function approximation with those offered by the SOM, with respect to RL and continuous action spaces.

8.2.1 The algorithm

The general problem with learning an optimal mapping from inputs to outputs using backpropagation is that the optimal outputs need to be known beforehand so that the appropriate errors can be fed back through the network. However, all that is known in figure 8.1 is the estimated expected reward of the *proposed* action, which is of no immediate help in learning to produce the *optimal* action. To circumvent this problem, the following algorithm applied to the architecture of figure 8.1 is proposed:

At each time-step, an input vector, I, is presented to the actor network and an output value, O, generated. O is then randomly perturbed[1] to produce O', and then this action is taken yielding a reward, R, from the environment. Meanwhile, the critic is also presented with the input vector, I, and generates the estimated expected reward of the pair (I,O), which we can call $E(R)$. If $R > E(R)$ then the random perturbation appears to have been successful and the actor network is updated towards[2] the pair (I,O'),

[1] In the same way as before — i.e. by adding random noise drawn evenly from the distribution, $[-\sigma,+\sigma]$, where σ is the exploration rate (corresponding to *MA* in the proposed model).

[2] Using the standard backpropagation learning rule (Rojas (1996)(chapter 7)) with I as the input vector and O as the target.

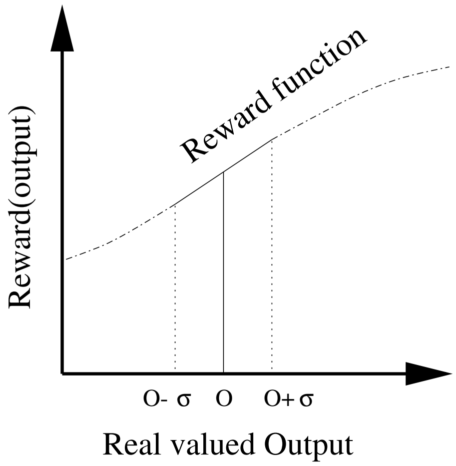


Figure 8.2: The reward function for a given input is linear in the real-valued action that is taken, providing σ is small.

otherwise the actor is updated towards (I,O) (in other words no update is made since the error is zero). Finally, the critic is always updated towards the pair (I,R), so that the estimated expected reward of the pair (I,O) will be more accurate next time it is required.

An intuitive validation of this technique can be achieved by considering a single input presented over and over, with perturbations to the output made in the range $[-\sigma,\sigma]$, and the environment yielding immediate non-discounted reward. If σ is small enough, then the reward function (assuming it is c(1) continuous) will approximate a straight line in the range $[O-\sigma,O+\sigma]$ as depicted in figure 8.2, and therefore the average environmental reward under a network proposing output O will be the expected environmental reward of *actually* taking action O, in spite of the network's random exploration around O. The critic should then produce an accurate estimate of the expected value of taking action O, again in spite of the exploration. Now, since the actor network is only updated towards actions that are better than this value, the actor network will tend to improve its performance. In practice, larger values for σ may still yield good results.

8.3 Gullapalli's SRV unit

Before this model is evaluated and compared with the proposed model of chapter 5, we first compare its performance with that of SRV, since this is the existing algorithm to which it bears closest resemblance, and also by which it was inspired. The SRV algorithm of Gullapalli (1990) is first recapitulated (from section 4.9).

Recall that the problem that Gullapalli sets himself is exactly the one considered here — namely that of generating an optimal mapping[3] from real-valued input vectors to a real-valued output guided only by a scalar reward signal. His approach is based upon what he calls the *SRV* ('Stochastic Real-Valued') unit of figure 8.3. The system is fed two inputs (the model can be generalised to any number of inputs) and an output O is generated using a Normal distribution. The mean for this distribution is generated by a single hidden unit, $H1$, whose output is the weighted sum of its inputs (including a bias). The second hidden unit, $H2$, which is also a linear unit, generates the estimated expected reward $E(R)$ of the SRV unit for the given input stimulus.

Gullapalli restricts the reward signal to the range [0,1], so $E(R)$ also lies within this range. The variance of the Normal distribution is generated according to $E(R)$ and the following principle; if $E(R) = 1$ then the unit is behaving optimally, and the variance is zero. If $E(R) = 0$ then the SRV unit is behaving very poorly and variance is set high. The variance is actually calculated from $E(R)$ using a monotonically decreasing function, $S(x) = max(\frac{1-x}{5}, 0)$. After an input vector has been presented to the SRV unit, and the action generated by the Normal distribution has been taken, a reward R is elicited from the environment. Now hidden unit $H2$ learns to produce R for the current input vector by backpropagating the error $R - E(R)$ through the linear unit according to the Widrow-Hoff learning rule (Widrow and Hoff, 1960):

$$u_i(t+1) = u_i(t) + \beta(R - E(R))I_i(t) \tag{8.1}$$

[3] i.e. That which maximises expected reward.

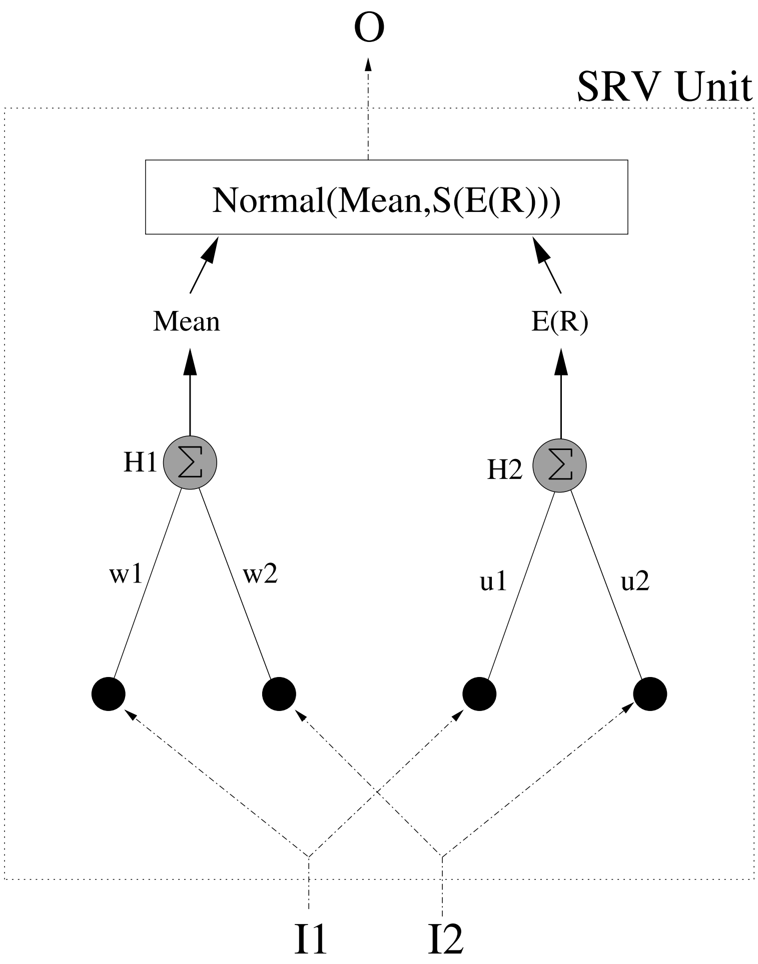


Figure 8.3: Gullapalli's SRV unit.

This is just a special case of the more general backpropagation update rule where each weight is updated negatively proportionally to the gradient of the Error function with respect to that weight. Here the error function is implicitly defined as $\frac{1}{2}(R-E(R))^2$.

Unit $H1$ is updated in much the same way except that its target is $O \times (R-E(R))$. Hence if the actual reward is an improvement on the estimated expected reward, then the mean of the variance is pushed towards O (since the exploration was probably good), otherwise the mean is pushed away from O. The actual update rule used for unit $H1$ is:

$$w_i(t+1) = w_i(t) + \alpha(R - E(R))\left(\frac{O(t) - mean(t)}{\sigma(t)}\right) I_i(t) \tag{8.2}$$

The intuitive idea behind the SRV is simple. For each input vector a mean and variance is computed, with the variance linked to how far the estimated expected reward is from a theoretical maximum. An output is then generated according to a Normal distribution based on this mean and variance. If the output is good then the mean is pushed towards this output otherwise it is pushed away from it. In any case, the estimate of the expected reward is updated. Learning proceeds in this way until the SRV unit produces the correct output for each input, and the variance of the distribution is therefore zero.

Gullapalli tests a single unit on the AND problem:

I1	I2	Output
0.1	0.1	0.1
0.1	0.9	0.1
0.9	0.1	0.1
0.9	0.9	0.9

with the reward at each time-step defined as $1 - |Error|$ where the $Error$ is the difference between the target output and the actual output of the SRV unit. The single unit easily learns the mapping in around 2000 time-steps, where a time-step corresponds to a single processing cycle involving: the presentation of a single input vector, taking the action proposed by the SRV unit, an environmental reward, and the application of the weight update rules. However, more interesting is the linearly inseparable XOR problem:

I1	I2	Output
0.1	0.1	0.1
0.1	0.9	0.9
0.9	0.1	0.9
0.9	0.9	0.1

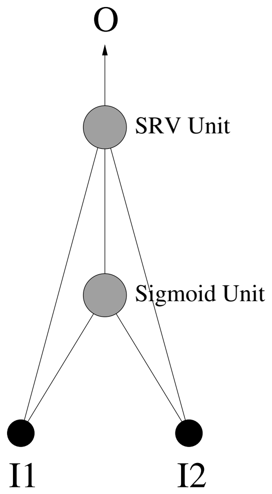


Figure 8.4: An SRV unit and a standard sigmoid unit are connected in a topology that is often used to solve the XOR problem.

Clearly the basic SRV unit will not be able to learn this mapping since the two hidden units, $H1$ and $H2$, only have a single layer of weights each. To solve this problem Gullapalli experiments with several ways of building networks of SRV units. The most successful involves using standard sigmoid units in a hidden layer and SRV units in the output layer. The exact architecture for which results are presented is replicated in figure 8.4. Gullapalli reports poor performance and unreliable convergence. A reproduction of the learning curve averaged only over those runs that converged is shown in figure 8.5 and labelled 'Standard Reward'. Gullapalli improves this preliminary result by augmenting the reward signal to include a term which provides positive reward as long as the outputs generated by the input vectors $\langle 0.1, 0.9 \rangle$ and $\langle 0.9, 0.1 \rangle$ are higher then the outputs generated by the other two input vectors. In this way, the behaviour is *shaped* with the network first attempting to gain partial reward for solving the weaker XOR problem, and then trying to optimise the reward by solving the full problem. The averaged error curve, which is shown in figure 8.5 and labelled 'Shaped Reward' is an improvement, although the network still fails to converge on all trials. Gullapalli's experiment is not repeated here, and the results are taken directly from Gullapalli (1990).

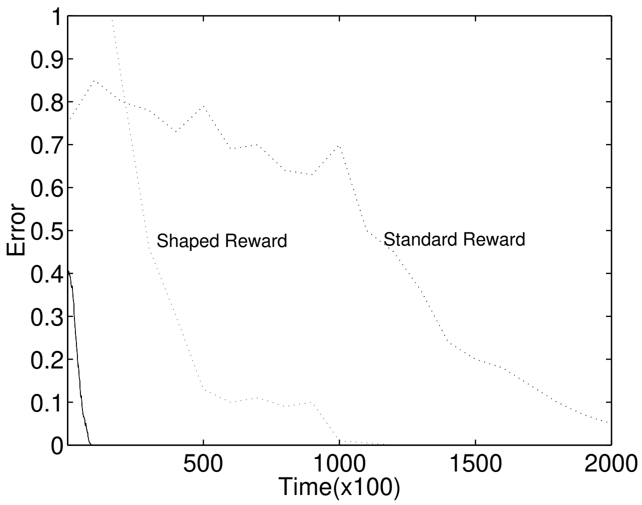


Figure 8.5: Learning curves for the XOR problem. The Error is the average positive distance between the network output and the target output. One time-step consists of presenting the network with one of the four input patterns and invoking one backpropagation cycle. The curve labelled 'Standard Reward' is Gullapalli's first attempt. Not all runs converged (6 out of 20 either did not converge at all or converged to a sub-optimal solution) and this curve is averaged over only those runs that did. The plot labelled 'Shaped reward' utilises a more informative reward signal (see text) in which learning is faster, but still only 17 out of 20 runs converged to 'correct weights'. The solid line is the performance (averaged over twenty runs) of the actor-critic model of figure 8.1 in which convergence occurred on every run (see following text for experimental setup).

8.4 Actor-Critic vs SRV

The real-valued actor-critic model of figure 8.1 is now tested on the XOR problem. The networks used for the actor and the critic are identical and consist of two input units, one output unit, and a single layer of six hidden units. It was noted that too few hidden units resulted in some runs failing to converge; at least three hidden units appeared to result in reliable convergence. The output units compute the sigmoid function, and the hidden units compute the *tanh* function (an empirical choice, with *tanh* outperforming the sigmoid in this particular experiment). In each time-step one of the four input vectors (chosen at random, always with equal probability), I, is presented to both the actor and critic networks, and the outputs O and $E(R)$ are generated respectively. O is perturbed by an amount drawn randomly and uniformly from the interval $[-\sigma, \sigma]$ to produce O' and this action is taken, eliciting an actual reward, R, from the environment. As in Gullapalli's first experiment, $R = 1 - |Error|$, where the Error is the difference between the actual output and the desired output. If $R > E(R)$ then the actor network is trained towards the pair (I, O'), otherwise the actor is not changed. The critic is always trained towards the pair (I, R).

8.4.1 Experimental setup

Before the result of the actor-critic model applied to the XOR problem is presented, some comments on algorithmic efficiency and parameter settings are called for in order to clarify the experimental setup.

One of the key criticisms repeatedly levelled at backpropagation is that it is slow to learn. The problem arises from the gradient of the Error function being very small with respect to one or more weights, particularly those lower down in a network. Since weight updates are made negatively proportionally to this gradient, convergence to local minima can be arbitrarily slow when the network is in a wide and shallow error basin. Problems also arise when the direction of the gradient is not directly towards the local minimum. The consequence of this is that many orthogonal oscillations may be required, with each update only resulting in a small component in the actual direction of the minimum (Rojas (1996)(chapter 8)).

Many extensions to the classical backpropagation algorithm have been suggested in order to address these issues. For example, in the *Delta-bar-delta* algorithm (Jacobs, 1988) each weight maintains its own learning rate which is increased if successive updates for that weight have the same sign, and decreased otherwise. This can both increase the speed of movement across shallow regions of the error surface and ameliorate the problem of oscillating orthogonal updates. *Rprop* (Riedmiller and Braun, 1993) employs a similar concept, while the *Quickprop* algorithm (Fahlman, 1989) proposes approximating the Error surface as a quadratic function which can be estimated using two successive error values, and then using this approximation to move more quickly towards the minimum. More sophisticated methods also exist such as the *conjugate gradient algorithm* (see Bishop (1995) for a description) which aims to discover and exploit the actual direction of the minimum using second derivative gradient information.

The number of different flavours of backprop is large and comparison between different algorithms on benchmark tests seems to be rare. Some improvements to the basic algorithm work well for some examples, but particular counter-examples can often be found (Rojas, 1996)[pg. 183]. There also tend to be costs associated with more advanced techniques such as an increase in the number of parameters (Delta-bar-Delta for example), the need for extra fixes to make the algorithm work in practice (Quickprop), or extra calculations such as those required for methods which utilise the Hessian matrix of second derivatives. The interested reader is referred to Bishop (1995) and Rojas (1996) for comprehensive and authoritative reviews.

Notwithstanding this discussion, classical backpropagation is used here, although a simple and effective way to speed up the learning of the XOR function turns out to be to use a high learning rate. A value of 0.5 works well, and a further speed up factor of 5 applied to the updates of weights in the hidden layer was also empirically beneficial. This presupposes that the global minimum lies in a suitably wide and flat region of the Error function. Although this approach produces good results in this particular task, one of the more sophisticated methods mentioned above is likely to be preferable in the more general case. It is also likely that the results presented here could be improved further with the application of one of the above algorithms.

An exploration size of $\sigma = \frac{1-E(R)}{2}$ is used which is analogous to Gullapalli's idea of increasing the exploration with the distance of the action from optimality. Necessarily, this assumes that the optimum reward is known before hand, that optimal behaviour is attainable (to avoid residual noise), and that the reward function is such that a feedback loop that will increase the noise out of control is avoided. In the general case, such assumptions are clearly unjustified, but in this specific case, and for the purposes of comparison, they are deemed appropriate. However, it is noted that the system is sensitive to this parameter. For example, setting the exploration to $\frac{1-E(R)}{4}$ produces slower convergence by around a factor of two.

Since the reward is received in the interval $[0,1]$, the sigmoid output units of the critic will find it increasingly difficult to accurately generate $E(R)$ as the behaviour approaches optimality. This is because the gradient of the Error function becomes very small for Outputs close to zero or one. For this reason the actual activation function

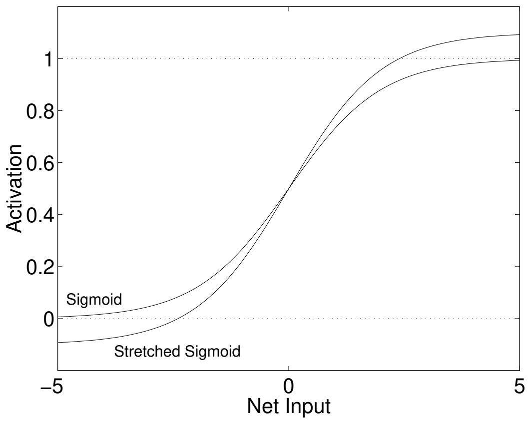


Figure 8.6: The usual sigmoid activation function, and the 'stretched-sigmoid' function used in the actor and critic networks to facilitate the networks' behaviour close to optimality. The stretched function is given by $y = \frac{1}{1+e^{-x}} \times 1.2 - 0.1$.

used for the output unit is slightly different to the usual sigmoid function, as shown in figure 8.6. A side-effect is that the backpropagation derivation is slightly different. Two alternatives would be to normalise the reward to a sub-range of $[0,1]$, or to use linear output units.

In the case of the XOR problem, the training data consists of four input patterns. The results presented here are achieved by following the convention of presenting the input patterns in random order (with each input pattern having an equal chance of being selected at each time step, i.e. with replacement). Contravening this convention appears to both reduce the proportion of runs that converge, and to increase sensitivity to parameter settings. This effect is not investigated further here, but represents a possible avenue for future work. A final detail concerns the initial weights of the networks. These are drawn randomly and evenly from the range $[-0.5, 0.5]$.

For clarity, the key parameters are summarised below. The values are identical for both the actor and critic networks.

Parameter	Value
Number of hidden units	6
Output unit activation function	'stretched sigmoid'
Hidden unit activation function	tanh
Perturbation range, σ	$\frac{1-E(R)}{2}$
Learning rate (weights to output layer)	0.5
Learning rate (weights to hidden layer)	0.5×5
Initial Weights	Evenly from the range $[-0.5, 0.5]$
Training data selection	Equal probability

8.4.2 Results

Having settled on an exploration strategy, the particular flavour of backpropagation algorithm, activation functions, and the parameters for the actor and critic networks, the learning curve for the XOR problem is shown alongside Gullapalli's results in figure 8.5. Convergence occurred ten times quicker than Gullapalli's best results, and on every trial.

This improvement by more than an order of magnitude is surprising since in essence the two approaches are very similar. Moreover, Gullapalli's XOR architecture is a specific instance of our model in which there is a single sigmoidal hidden unit, and two linear output units — one for the actor and one for the critic. Merging the actor and the critic into the same network (i.e. a network with two output units — one corresponding to the actor, the other to the critic) was not found to adversely affect the results, so a number of other possibilities remain for why performance is so much worse.

It is possible that there are an insufficient number of hidden units in Gullapalli's model. In practice, the critic network (or critic unit in the case of Gullapalli's model) may have to learn a number of different functions as the actor portion of the system experiments with different policies. This suggests that more hidden units may be required than would normally be expected to solve the XOR problem in a supervised fashion. In our actor-critic model, convergence only occurs reliably when the hidden layer contains at least three hidden units. Another possibility is that Gullapalli's learning rates are too small, unnecessarily delaying learning. Using half the learning rate and removing the speed-up factor of 5 in the hidden layer weight updates reduces the speed of convergence by a factor of about 10 in the actor-critic model. Another difference is the use of linear rather than sigmoidal units in the output layer. Interestingly, using linear output units in the actor and critic networks resulted in some runs converging to suboptimal performance[4]. It is also possible that the exact formula used for linking exploratory noise to the current estimate of expected reward may be a relevant factor, although this seems unlikely. One final discrepancy is the way in which noise is added to the proposed actions; Gullapalli uses Normally distributed noise, whereas in the actor-critic model we use evenly distributed noise.

8.4.3 Scalability

At this point we make a brief note about scalability. Although in this example only one output is required and the reward is immediate, it is noted that either of the models being compared could in principle be extended to incorporate multiple output variables (either on separate actor networks, or on a single network), and discounted sums of reward.

8.5 Actor-Critic vs proposed SOM-based model

Having made some effort to arrive at a suitable backpropagation architecture for the reinforcement learning of real-valued actions, we can now proceed to the main focus of this chapter — a comparison with the SOM-based model of chapter 5. We begin by presenting the XOR problem to the SOM-based model. Input and Output maps of 10x1 units are used, initial neighbourhoods are set to 5, and the learning rates are annealed in the usual way with $f(t) = 0.995^t$. Figure 8.7 shows the result. Note that the scale is smaller than that of figure 8.5 by a factor of twenty, and that learning proceeds several hundred times quicker than Gullapalli's comparable result. The problem is solved within a few hundred time-steps compared with a few thousand time-steps for

[4]Although it is always hard to detect the difference between the network converging to a local minima, and slow learning caused by an almost flat Error surface.

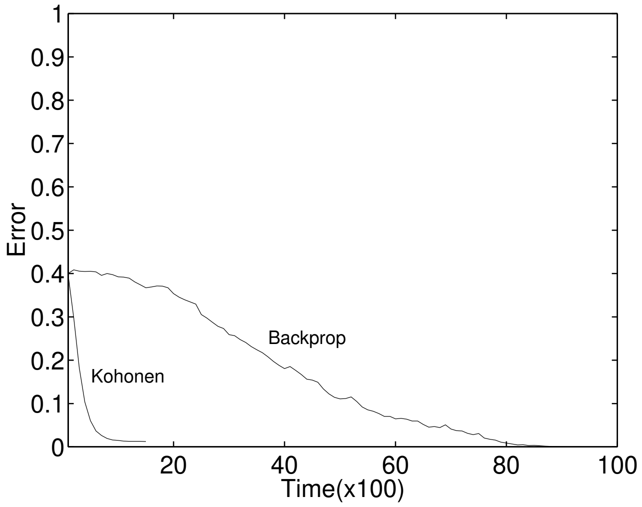


Figure 8.7: Learning curves for the XOR problem. The curve labelled ‘Backprop’ corresponds to the solid line in figure 8.5. The curve labelled ‘Kohonen’ shows learning in the SOM-based model with the annealing function, $f(t) = 0.995^t$, optimised by hand.

the backpropagation model and hundreds of thousands of time-steps for Gullapalli’s SRV network. As with the backpropagation model, convergence occurred on 100% of trials.

The Kohonen model appears well suited to this problem since it is able to take advantage of the discrete nature of the task. There are only four singularities in the input space that need to be represented (corresponding to the four input patterns) and this can be achieved very quickly and reliably with a Kohonen map. Similarly, there are only two points in the output space that need to be identified, and this is also easily achieved. No interpolation between these points is required, and so no generalisation is required either. This experiment is not analysed in any more detail because the result is both very clear and very easy to reproduce.

8.5.1 Learning to throw projectiles

A more interesting test for the actor-critic model requires the learning of a continuous function, as in the projectile throwing problem of the previous chapter. We might expect that the backpropagation-based model is able to take advantage of the fact that continuous functions are built directly into its representational machinery, while we know that the SOM-based model must use a discrete approximation (as already seen in figure 7.6 for example).

By parameterising the problem on the position of the target, the wind speed, the coefficient of friction and the mass of the projectile, we can also test both models in a higher dimensional state-space than considered so far. In particular, we would like to see how the SOM-based approach scales to a problem with an intrinsic dimensionality significantly greater than the map itself, and also how much more robust, if at all, the actor-critic model is to an increase in the dimensionality of the problem.

Four experiments are considered in this section. In the first, only the position of the target is varied, leading to a single dimensional state-space and a problem similar to that encountered in chapter 7. In the second experiment, the wind speed is also varied resulting in a two-dimensional state space. The third experiment adds a third variable — the coefficient of friction — and the final experiment adds a variable projectile mass. Due to the possible interactions between the parameters, care must be taken to ensure that the target is never outside the throwing range of the agent. Table 8.1 summarises the four experiments.

Experiment	**Distance**	**Wind**	**Co. Fr.**	**Mass**	**No. Variables**
A	$[0,9]$	0	0.1	1	1
B	$[0,9]$	$[-3,0]$	0.1	1	2
C	$[0,9]$	$[-3,0]$	$[0,0.2]$	1	3
D	$[0,9]$	$[-3,0]$	$[0,0.2]$	$[1,2]$	4

Table 8.1: Four experiments involving different numbers of variables, with each variable selected randomly and evenly from within a specified range. Both variable ranges and, where appropriate, constant values are indicated. Care must be taken to ensure that the target can always be reached (in principle) from anywhere within the input space.

The reward is given in the usual way — i.e. according to equation (7.4) — normalised to lie in the range $[0,1]$[5]. However, before the experiment proper is introduced, we first report a problem that was initially encountered with this reward function. Preliminary results showed very little difference in performance across the four experiments,

[5]Normalised reward = $max(\frac{r}{20},-1)+1$.

even when the variables were *not* supplied as part of the input. An inspection revealed that the system, to avoid the errors caused by incorrectly predicting the effect of the environmental parameters, was consistently favouring extremely hard, low throws. By minimising the time the projectile was in flight, solutions were being found that corresponded to the flattest regions on the reward surface (that still maximised reward). This finding confirmed a previous intuition that given a choice of equally good solutions, those that are most reliably good will tend to be preferred. This is a straightforward consequence of aiming to maximise expected reward, and is quite reasonable behaviour. However, since the aim here is to compare the performance over *non-trivial* reward surfaces of various dimensions, the reward function was adapted to artificially penalise hard, low throws. This was achieved by simply subtracting two from equation (7.4) if the vertical component was less than the horizontal component. This lowers the favoured flat region of the reward surface and effectively dictates that each throw should be at least 45° above the horizontal.

The experimental setups of the two models are not laboured again, and are summarised directly in tables 8.2 and 8.3. The key change to the actor-critic model is that now the actor network has two output units, one for each component of the throwing velocity. Note that the actor-critic links the exploration to the estimated expected reward in exactly the same way as before which conveniently removes the need for a manual search for an optimal annealing rate. However, such a manual search is still required for the SOM-based model because it is not clear how to integrate the exploration, p, of different action units (the exploration associated with standard Q-learning) with the expected reward. A complete exploration of the models' parameter spaces is outside the scope of this experiment; hence the following values represent the best found during a small amount of off-line trial and error.

Parameter	Value
Input map size	10×10 units
Motor map size	30×1 units
Input map neighbourhood size, IN	$7 \times f(t)$
Motor map neighbourhood size, ON	$7 \times f(t)$
Reward horizon, h	1
Q-learning rate, α	$f(t)$
Learning rate of Input map, IL	$0.3 \times f(t)$
Learning rate of Motor map, OL	$f(t)$
Probability of Q-learning Exploration, p	$f(t)$
Probability of Large Motor Exploration, q	$f(t)$
Max. Exploration distance around Motor unit, MA	$f(t)$
Annealing schedule, $f(t)$	0.9997^t

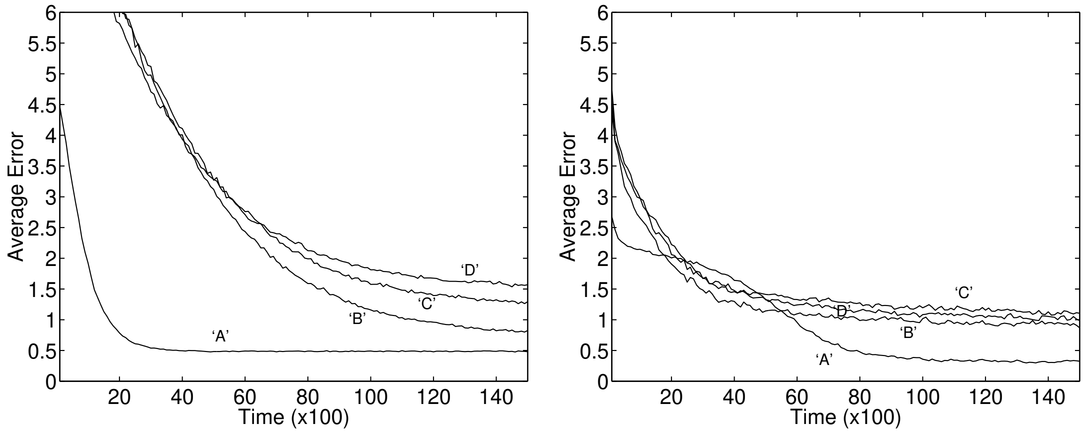
Table 8.3: The SOM-based model parameters for the four experiments of table 8.1. Experiment 'A' exceptionally used a single dimension Input map (30×1, neighbourhood$= 7 \times f(t)$) and was able to utilise a faster annealing schedule, $f(t) = 0.9985^t$.

Parameter	Value
Number of hidden units	8
Output unit activation function	'stretched sigmoid'
Hidden unit activation function	tanh
Perturbation range, σ	$\frac{1-E(R)}{2}$
Learning rate (weights to output layer)	0.25
Learning rate (weights to hidden layer)	0.25
Initial weights	Evenly from the range $[-0.5, 0.5]$
Input data selection	Independent and evenly distributed

Table 8.2: The actor-critic model parameters for the four experiments of table 8.1.

Results

Figure 8.8 compares the results of the two models for the four experiments. The first observation is that adding extra variables reduces the performance of both models. The single exception is the application of the actor-critic to experiment D in which a modest improvement is observed. This is likely to be because when the mass is varied,



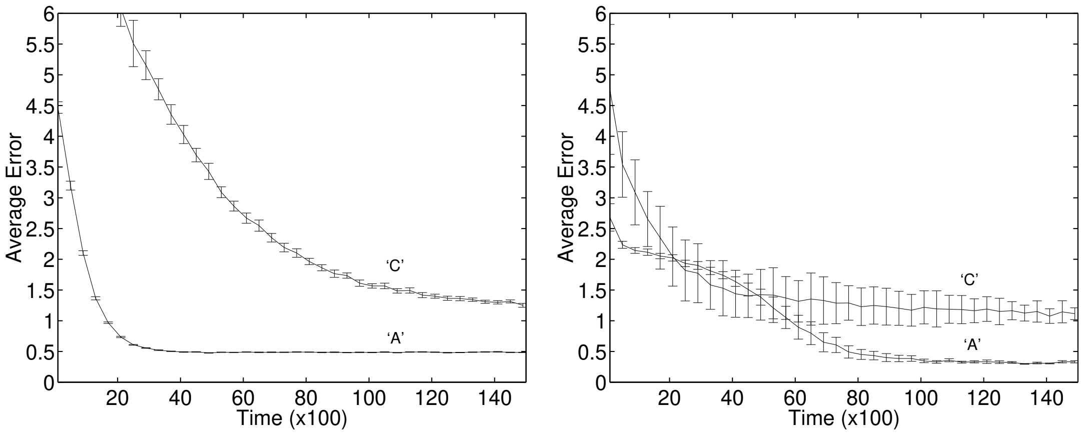
(a) Learning in the SOM-based model.

(b) Learning in the actor-critic backpropagation model.

Figure 8.8: Comparison of Error curves for learning to throw a projectile to a target in the two models. The Error is simply the average distance from the target over the last 100 time-steps. Additionally, each curve is averaged over 100 independent trials, so each data point actually corresponds to $100 \times 100 = 10000$ individual throws. Each graph contains four plots which correspond to the environment parameters of table 8.1. Each curve represents the best set of parameters found for that experiment. The standard deviation of the mean is negligible which means error-bars, if shown, would be hardly visible. However, for completeness, variances of some curves (the data, not the mean) are shown in figure 8.9.

its new mean is higher than when the mass is fixed. A higher mass effectively reduces the impact of the wind and friction because forces applied to the projectile are divided by the mass to yield the acceleration.

Looking first at the results for the actor-critic model, we see that each of the experiments of two or more input variables perform very similarly suggesting a robust response to higher dimensional spaces. This is in contrast to the same results for the SOM-based model which indicate that this approach is more susceptible to the curse of dimensionality. This is to be expected since in experiment 'D', for example, the two-dimensional Input map must represent an intrinsically four-dimensional space. We therefore expect over-generalisation to occur, as well as a reduction in the efficiency and appropriateness of neighbourhood Q-learning. Increasing the dimensionality of the Input map is not a general solution since the number of units required to create an appropriate map will become very large, as will the amount of training data.



(a) Variances of 'A' and 'C' from figure 8.8(a). The variance of 'C' is typical of 'B' and 'D' also.

(b) Variances of 'A' and 'C' from figure 8.8(b). The variance of 'C' is typical of 'B' and 'D' also.

Figure 8.9: Typical variances of the data (not the mean of the data — see appendix C for the difference) for the graphs of figure 8.8.

However, when the input space is of low dimensionality, we note good performance by the SOM-based approach. In particular, note that in experiment 'B' (involving two variables), a two-dimensional Input map performs as least as well as the actor-critic in terms of final performance. In this case, a two-dimensional map is used in a two-dimensional input space so we would expect performance to be good. In experiment 'A', involving just a single dimension state space, the SOM-based approach is able to learn much quicker than the actor-critic as a result of using a smaller one-dimensional Input map. It is not clear why the actor-critic's response to experiment A has a different signature to the other graphs.

One point worth noting is that although the performances of the two models appear quite similar in terms of final performance, the difference between an average error of one unit along the x-axis, and an average error of two units may be more significant than it would first seem. As an illustration, imagine that the effects of each of the four environmental variables were arbitrarily large on the landing position[6], and that

[6]Although the target position cannot affect the landing position of a trajectory, so we are only really concerned with the other three environment variables here.

the system was given no information whatsoever before each throw. Under these circumstances the system could (and would be expected to) make each throw with a zero velocity, guaranteeing a landing position at the origin. Given that the target position was randomly generated in the range $[0,9]$, this would result in a minimised expected error of 4.5. Hence we would never expect the system to achieve an average reward of less than -4.5, no matter how poor it is in differentiating between different input vectors. In reality, because the effect of the environmental variables will not be this severe, an upper bound on the average error of less than 4.5 is expected. The point is that an average error of around two shows that the system has learned to differentiate between different environmental variables probably about as poorly as it could possibly be expected to do, even though some of the error curves start much higher.

The conclusion of this comparison, and the previous XOR experiment, is that for low dimensional problems, the SOM-based model is both fast and effective. However, its performance soon deteriorates as the intrinsic dimensionality of the input space is increased to beyond the dimensionality of the Input map. In contrast, the actor-critic model, based on backpropagation, appears to scale better with respect to the dimensionality of the problem both in terms of learning speed and final performance.

8.5.2 Higher dimensional input spaces

The conclusion of the previous section is reinforced by considering a further, and very simple experiment in which the reward surface is easy to visualise, even in a very high dimensional input space. The idea is to *hide* a single relevant input variable amongst a number of other variables, with none of these other dimensions having a bearing on the reward function.

Let us first consider the problem of mapping a single input variable, drawn randomly and evenly from the range $[0,1]$, to a single output variable, with the desired output being equal to the current input. Following previous experiments, we can use an immediate reward signal given by:

$$r_t = 1 - |output_t - input_t|$$

In this way, we expect the system to learn to map the single input variable onto itself. Now we can increase the number of input variables, with every variable drawn randomly and evenly from the range $[0,1]$. The goal is still to produce a single output that mirrors the first variable (reward given as before), but now the system must extract the importance of this single variable from the other input dimensions, all of which represent noise. Naturally, the system is not explicitly told which the significant variable is.

This task was performed by both the actor-critic and SOM-based models for a range of input dimensionalities. The experimental parameters are summarised in tables 8.4 and 8.5, and the results are shown in figure 8.10. Although no special effort was made to optimise performance in terms of learning time, the graphs are considered to be representative of the relationships between the best obtainable learning curves. As in the previous section, optimising the SOM-based model requires searching for the best annealing schedule that does not compromise final performance, and optimising the actor-critic model requires discovering a suitable learning rate. Note that learning in both models could be achieved faster in low dimensional input spaces, as suggested by the results of the previous section for example. However, the annealing schedule of the proposed model, and the learning rate of the actor-critic, were adapted for the higher dimensional cases and no attempt was made to independently optimise these parameters for the lower dimensional cases.

Parameter	**Value**
Number of hidden units	2
Output unit activation function	'stretched sigmoid'
Hidden unit activation function	tanh
Perturbation range, σ	$\frac{1-E(R)}{2}$
Learning rate	0.05
Initial weights	Evenly from the range $[-0.5, 0.5]$
Input data selection	Independent and evenly distributed

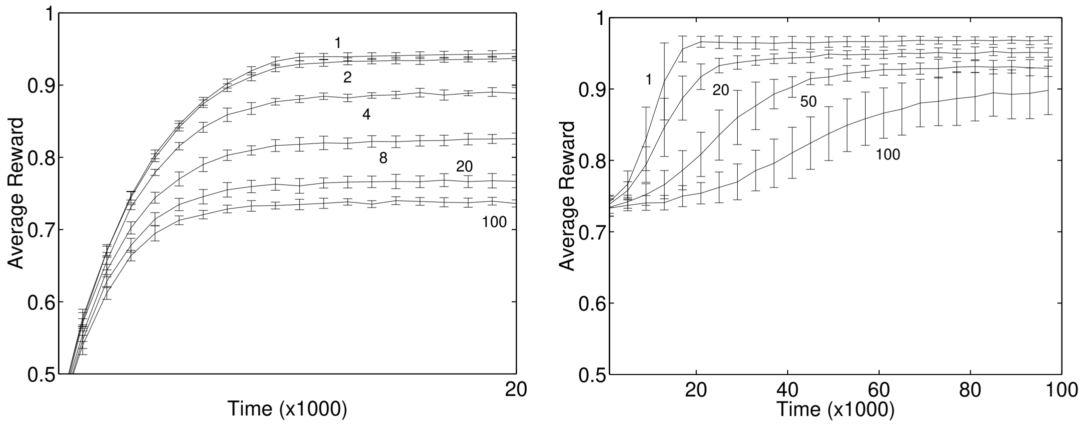
Table 8.4: The actor-critic model parameters used to obtain the results of figure 8.10.

Parameter	Value
Input map size	20×20 units
Motor map size	30×1 units
Input map neighbourhood size, IN	$7 \times f(t)$
Motor map neighbourhood size, ON	$7 \times f(t)$
Reward horizon, h	1
Q-learning rate, α	$f(t)$
Learning rate of Input map, IL	$0.03 \times f(t)$
Learning rate of Motor map, OL	$f(t)$
Probability of Q-learning Exploration, p	$f(t)$
Probability of Large Motor Exploration, q	$f(t)$
Max. Exploration distance around Motor unit, MA	$f(t)$
Annealing schedule, $f(t)$	0.9997^t

Table 8.5: The SOM-based model parameters used to obtain the results of figure 8.10.

As we might expect given the results of the previous section, the SOM-based model scales considerably worse with the dimensionality, d, of the input space. In particular, good performance is maintained with up to $d = 50$ in the actor-critic model, whereas even four dimensions begin to cause problems for the two dimensional Input map of the proposed model. Note that we expect 0.75 to be a lower bound on performance, since given no input data at all, the system would in principle still be able to achieve this level of expected reward (average error = 0.25) by always producing an output of 0.5.

There are some other interesting things to note about these results. The first is that the SOM-based model is able to reach its optimal performance — such as it is — more quickly than the actor-critic (note the difference in scale). Note also that the variances of the performances of the SOM-based method are much smaller than those associated with the backpropagation model. This can also be observed in figure 8.9. From this we can deduce that the actor-critic model is more sensitive to initial conditions, whereas in the SOM-based model it is the fixed annealing schedule that determines the performance of the system during learning. We also note a significant difference in the number of free parameters in each model. For any input dimensionality, d, the SOM method has $200 \times d + 6030$ weights, while the actor-critic has only $(2 \times d + 4) + (2 \times d + 2)$.



(a) Learning in the SOM-based model using the parameters of table 8.5. Each separate plot is annotated with a number corresponding to the dimensionality of the input space.

(b) Learning in the actor-critic backpropagation model using the parameters of table 8.4. Each separate plot is annotated with a number corresponding to the dimensionality of the input space.

Figure 8.10: Comparison of reward curves for learning to distinguish one relevant dimension in a d-dimensional input space. Each data point is averaged over the previous 1000 time-steps, with each time-step corresponding to the presentation of one input vector drawn randomly and evenly from the d-dimensional unit hypercube. Additionally, each curve is averaged over 25 independent trials, so each data point actually corresponds to $1000 \times 25 = 25000$ individual time-steps. Each graph shows a number of different experiments for different values of d. The variances of the results are also shown on each plot.

It is easy to see why the actor-critic performs better than the proposed model in this context. The backpropagation algorithm adapts the weights from the non-relevant input variables of both the actor and critic to zero, so that after learning only the relevant input dimension affects the outputs. In contrast, the Input map has no corresponding means of extracting the important variable, and each unit of the map is obliged to consider every dimension equally. When the dimensionality of the the input space is large, the map's units are unable to form an effective representation of the relevant variable.

8.5.3 Non-stationary environments and non-equiprobable input distributions

Continuing the comparison of backpropagation with the SOM-based model, the issue of dynamic environments is now considered. A potentially useful property of any RL application is the ability to track a non-stationary environment. By maintaining some plasticity in the form of non-zero learning rates and exploration, an agent may adapt to changes in the reward function, changes in the input distribution, sensory-motor drift, internal or external damage, or allow for re-calibration. But these facilities all require that the underlying generalisation mechanism can support such adaptation.

The following section is intended to be of a qualitative rather than quantitative nature and specific model parameters are not discussed. The essence of the following results is not parameter critical, and could be reproduced with a range of values drawn from the previous discussions of the two models.

Let us begin by considering the function of one variable shown in figure 8.11(a), in which reward is given according to the negative of the distance between the desired output and the actual output. This simple mapping can easily be learned by both the actor-critic and SOM-based models. As we have seen, in the case of the backpropagation method, the problem is one of adjusting the weights of the actor to approximate the function. In the SOM method, the input and output spaces have to be explicitly represented by units of two Kohonen maps, and then a mapping between these discrete states approximated through any appropriate value estimation technique.

In order to simulate a non-stationary environment, half way through training the task is altered. After 20000 time-steps, rather than the input being evenly distributed in the range $[0,1]$, it is now distributed evenly in the subrange $[0,0.5]$ so that the system no longer has exposure to the second half of the problem. In addition, the desired mapping in the first half of the function is changed so that the target is always zero. The new state of affairs is shown in figure 8.11(b). The question now is how do the two methods adapt to this significant and sudden environmental change.

To answer this question, the results of running the experiment on the two algorithms

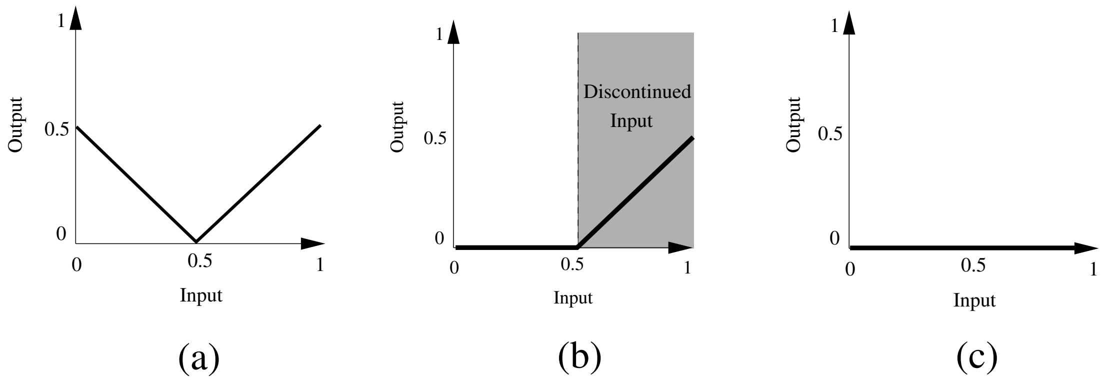


Figure 8.11: Figure (a) shows a mapping from one input to one output calculated by $y = |0.5 - x|$. In the first half of training, the input variable is randomly and evenly distributed in the range $[0,1]$, but, after 20000 time-steps, input in the range $[0.5,1]$ is discontinued and the desired mapping in the range $[0,0.5]$ is changed to $y = 0$ as shown in (b). (c) shows the new function learned by the backpropagation algorithm where a side effect of learning the new mapping in the range $[0,0.5]$ is interference with the learned mapping in the range $[0.5,1]$.

are shown in figures 8.12 and 8.13. Figure 8.12 shows the effect of changing the environment on the backpropagation model in the way described above. Curve A shows the average error (negative of reward) per time-step of the system. The network immediately learns to approximate the initial mapping to within an error of about 0.15, and then proceeds to slowly refine its performance[7] until the environment is abruptly changed at $Time = 20000$. At this point, the problem is reduced to the much simpler one of producing a zero for every input (in the subrange $[0,0.5]$), and performance reflects this by quickly approaching optimality.

However, the point of interest is observed in the degradation in the learned mapping of the *discontinued* half of the input space. The network is simultaneously run (but not trained) on inputs selected randomly and evenly from the interval $[0.5,1]$. The average error of the network for this part of the environment is shown as curve B. Initially, the error is predictably similar to the actual error received by the learning system.

[7]Interestingly, learning is considerably slower than in the single dimensional projectile experiment of figure 8.8. We believe that this is due to the non-monotonic nature of both the mapping and the reward function. This relates to linear-inseparability in that if we were to take three input-output pairs from the desired mapping of figure 8.11(a) at $\langle 0,0.5\rangle$, $\langle 0.5,0\rangle$ and $\langle 1,0.5\rangle$, and label them 'class 1', 'class 2' and 'class 1' respectively, we would have a linearly inseparable problem. Non-monotonic regression problems were found to be harder to learn than monotonic ones with backpropagation, either directly or through the RL approach of the actor-critic.

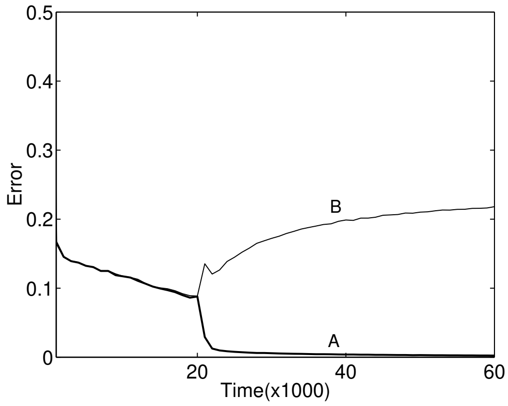


Figure 8.12: Adapting to a dynamic environment with the backpropagation method. Curve A shows the actual error while curve B shows the error that would be incurred by the network with respect to the discontinued half of the distribution. The environment is changed at $Time = 20000$. The error at each time-step is the negative of reward. Results are averaged over a number of independent trials.

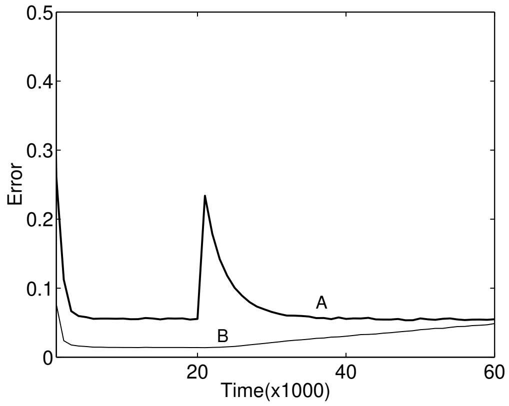


Figure 8.13: As for figure 8.12 but for the SOM based model.

However, at the point when the environment is changed, this error quickly increases as the network concentrates on the function in the range $[0, 0.5]$ at the cost of interference to the function in the range $[0.5, 1]$. Figure 8.11(c) shows the new function mapped by the network after the environmental change, and here the reason for the degradation in the performance of the actor network in the discontinued range is explained. The adaptation of the weights to fit the function in the range $[0, 0.5]$ has caused the network to produce an output of zero in the remaining part of the input space too.

Whether this behaviour represents over-generalisation or not clearly depends on whether the discontinuation of the input space in the range $[0.5, 1]$ turns out to be temporary or permanent. If the discontinuation turns out to be permanent then this interference could be useful, since the system is effectively forgetting something which is obsolete and for which, eventually, resources may no longer be available. But if the agent is just experiencing a local environmental change then the system is over-generalising and throwing away useful and possibly hard-earned information. Deciding whether the system is behaving sensibly or wastefully will require a knowledge of the environment.

It seems possible, and even likely, that a real-world environment will behave in exactly this way. After all it only took a thousand time-steps for the disruption to the discontinued part of the environment to occur, and this level of discontinuation would surely be expected in many applications. The ability of a system to preserve information acquired about a *temporarily* unexposed part of the environment is clearly important.

Figure 8.13 shows the same graphs for the SOM-based model. Recall that graph 'A' represents the error of the actual system as it learns first the mapping of figure 8.11(a) and then the mapping of 8.11(b), and graph 'B' represents the error in the mapping of the environment only in the discontinued range, $[0.5, 1]$. After $t = 20000$, the performance in graph 'B' is hypothetical, since the actual inputs no longer fall within this range.

The first observation is that learning is much quicker in the early stages[8]. The point of interest however is what happens after the environment is changed. Firstly we see a sudden drop in performance of the live system which slowly improves as the agent learns the new function of figure 8.11(b). The second thing to note here is that adaptation to the new environment is slower. This is partly because the residual exploration is small at this stage (maintaining larger residual exploration is costly in terms of performance), but also partly because the backpropagation network has fewer parameters and can change them all simultaneously in response to the new and very simple reward surface. In contrast, the SOM-based model can only make local changes to the mapping in one part of the input space at a time. However, being able to make changes locally becomes an advantage when it comes to preserving learned behaviours. This is witnessed in the second half of curve B which deteriorates much slower than the corresponding curve for the backpropagation model. Local changes can be made to the part of the environment that has changed without disrupting the remaining parts of the system. Such disruption only occurs as a result of neighbourhood activity in the Input and Output maps slowly dragging units away from their learned positions.

[8]The fact that curve B does not follow curve A prior to $t = 20000$ reflects the residual exploratory noise that has to be maintained in the system to allow adaptation to the expected change in the environment. The reason this exploratory noise is not necessary in the actor-critic is that exploration is again linked to reward following Gullapalli's original approach. However, note that in general this convenience will not be afforded to the actor-critic model either, because it relies on the optimal reward being known in advance.

The degree to which the tradeoff between plasticity and stability is made in the SOM-based model can be controlled either as a function of time or dynamically controlled as a function of an environmental variable such as expected reward. However, in the backpropagation model there is no general way of controlling this kind of global disruption resulting from local environmental changes since a distributed representation is a fixed feature of this class of models. Although this feature often turns out to be an asset, in this particular instance it also manifests itself as a potential liability[9].

On the more general subject of non-equiprobable input distributions, it is generally accepted that backpropagation can be sensitive to the ordering and distribution of the training data. This was observed while using the actor-critic to learn the XOR problem back in section 8.4. There, it was found that random presentation of the four input vectors considerably improved performance, and that contravening this convention resulted in runs that did not converge. Although temporarily discontinuing entire regions of the input space represents an extreme set of circumstances, it seems likely that more subtle shifts in training order or distribution could have similarly disruptive effects on the representations of the actor-critic model.

8.5.4 Further comparisons

We now compare some more general features of the actor-critic and SOM-based models.

Difficult functions

Some functions appear to be harder to learn than others using backpropagation. Consider the mapping of figure 8.14 which appeared in section 5.6.7. Following the brief discussion of non-monotonic functions in the previous section, we expect the highly oscillatory (and discontinuous) nature of this function, to make it very difficult to learn. Even learning such a function within a supervised framework where the input-output

[9]Li (1999)(pg. 27) briefly discusses this problem with respect to backpropagation. He goes on to review a number of alternative architectures, all of which address the problem by utilising multiple networks, with each network having a local scope.

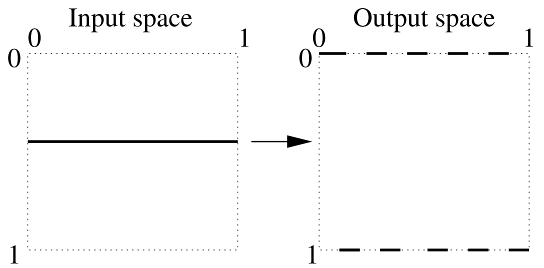


Figure 8.14: A non-contiguous mapping duplicated from section 5.6.7

pairs are provided explicitly may take a prohibitive length of time. Indeed this was found to be the case in preliminary experiments (see appendix E for the MATLAB script that was used) in which a variety of hidden layer sizes and learning rates were used. In these experiments the mapping of figure 8.14 was never learned. In general, acquiring the same function by trial and error reinforcement learning would be expected to be even more problematic. Although faster and more sophisticated backpropagation learning strategies exist which may help in the case of some difficult mappings, it is noted that even advanced optimisation strategies can suffer from local minima problems or very slow convergence (Bishop, 1995, pg 183).

In contrast, and as seen in section 5.6.7, the SOM-based model has no difficulty with this type of mapping because fitting the function in one region of the input space does not seriously constrain the task of fitting the function elsewhere. Here, the oscillatory variations that cause problems for the backpropagation model can be fitted locally and independently of other variations by a representation that is more explicit.

The root of the problem for the backpropagation model appears to be that feature which also makes the XOR problem — or the more general *N-bit parity* problem — difficult. In terms of classification tasks, we are referring to linear inseparability, but in terms of this regression problem we are considering the more general feature that the smallest possible change in the input produces the largest possible change in the output. Bishop (1995)(pg. 88) suggests that real world learning problems are not likely to possess this feature, but in the context of learning robots for example, the idea of a threshold in the input space with each side of the threshold requiring very different behaviours or yielding drastically different reward for the same behaviour does not seem entirely implausible.

Local minima

A potential hazard of any parametric fitting based on gradient descent is that of local minima. It is hard to evaluate the exact cost of this risk within the context of the actor-critic model presented here, but sensitivity (in terms of convergence) to initial weight values was encountered during the course of the experiments of this chapter. The equivalent failure of the SOM-based model is that the map converges to a disordered state in which topology is not preserved. But it has already been discussed that this need not prevent the system from acquiring optimal behaviour, and in this sense the SOM approach may be considered more robust.

Generating alternative actions

The SOM-based model maintains estimates of the expected reward for every action in every state (according to the current representation of the state and action spaces), and can therefore propose a list of alternative behaviours for each situation, ordered perhaps according to desirability. This feature potentially increases robustness and would be useful for example if some other part of an agent's control system was to veto the recommended optimal action. For example, an obstacle could be in the way of a proposed trajectory, or physical damage could prohibit a certain behaviour. The actor-critic model proposed here makes no provision for the selection of alternative actions.

An illustration of this issue was encountered while training the SOM-based model to throw a projectile to a target back in chapter 7. It was noted that for any given target position, there was a continuous range of different throwing velocities that reached that target, from high lobs to more shallow but harder throws. As seen in figure 7.2(a), almost the entire range of possible solutions was generated. The actor-critic model offers no means of representing these alternative solutions.

Interpretation and diagnosis

One other consideration is that of interpretation. An accepted problem with feed-forward networks is that they operate as a black box, and extracting meaningful knowledge out of a trained system may be a significant challenge. Diagnosing behavioural failure may also be problematic for similar reasons. Being able to interpret the meaning of a model's parameters could potentially be useful. Imagine a robot explorer operating in an uncharted environment such as the surface of Mars. Being able to interpret the trained control system may be useful for learning about this environment, and improving the next generation of learning models. In the case of the proposed model, it may potentially be possible to estimate the probability of certain environmental features for example.

8.5.5 Generalisation

As compensation for all the negative points already discussed, the key advantage of the backpropagation model is the power of its generalisation capability. After all, the problem observed in figure 8.11(c) (involving hiding half of the environment) was essentially one of over-generalisation. The SOM-based model will find it increasingly difficult to deal with high dimensional input and output spaces, and even in low dimensional spaces, generalisation may be significantly poorer than in a distributed model. This is observed in both figure 8.8, which compared the two approaches applied to throwing projectiles under variable conditions, and figure 8.10, which was concerned with a simpler problem but in an input space of up to 100 dimensions. The indication is that as a result of insufficient generalisation power, the SOM-based model may require a prohibitively large number of weights (and quantity of training data) in input dimensions as low as three, particularly where the dimensions are uncorrelated.

In many environments the input space may have high dimensionality, even though only a small fraction of those dimensions are actually relevant to a decision making process. Reported in Bishop (1995), Hartman et al. (1990) performed experiments in which out of twenty input variables only one was correlated with the output. While a backpropagation network was shown to be able to ignore the irrelevant inputs and learn the desired mapping with a small number of hidden units, a radial basis function, which can be considered as belonging to the same [local] class of function approximators as the SOM-based model considered here, exhibited poor performance which improved only slowly as the number of basis functions was increased. The relevance of this result to the proposed model is confirmed by the experiments of section 8.5.2.

The implications for the SOM-based model are clear — each input line has equal weight, and relevant parts of the input vector cannot easily be disambiguated from noise. Versino and Gambardella (1996, 1995) make a similar observation during their application of the Motoric map[10] to the supervised learning of visual-motor coordination of a mobile robot. Their solution is to select the winning unit for a given situation in stages, with those dimensions of the input space deemed most important being evaluated first (generating a subset of candidate winning units).

8.5.6 Local vs non-local representation

The comparison of the two models considered here contributes to a somewhat broader argument. The Kohonen-based approach could be described as *local* in the sense that the desired mapping is represented explicitly with each weight having only a local scope. In the case of the feed-forward network approach there is a less explicit relationship between the free parameters of the model and the mapping which the model learns, with each parameter having a global scope and influencing all parts of behaviour. This approach could therefore be described as *non-local*, or *distributed*, with changes to any one part of the system affecting all other parts to some degree. Note also that the SOM-based generalisation imposes very little prior structure on the solution. This is why the problem of figure 8.14 presents no difficulty.

Many of the differences between the SOM-based and actor-critic models that have been discussed so far appear to be a direct consequence of this local/distributed model class distinction, and we therefore expect the discussion to be valid for any algorithms of the same class. For example, if a learning architecture was constructed that utilised an ART

[10]This work is actually based on the *Extended Kohonen map* of Ritter and Schulten (1987). However, this appears to be very similar to the *Motoric Map* of Ritter et al. (1990).

network, or K-means clustering, or a radial basis network (Moody and Darken (1989), reviewed in Bishop (1995)), we would expect it to exhibit similar properties to the SOM-based model when compared with the actor-critic. Hence the broader debate is not whether backpropagation itself represents a more suitable approach to the problem than, say, a Kohonen map, but more importantly whether local or distributed function approximators are likely to offer the kind of features that we desire. We will shortly discuss which kind of problems will be suited to which generalisation framework in the light of the experimental evidence that has been uncovered in this chapter.

8.6 Summary

Some serious drawbacks have been identified with both local and distributed models with respect to the class of problems considered in this thesis. The issue is resolved for the time being with a summary of the type of applications for which each model class is expected to be best suited.

The experiments suggest that the SOM-based model will have a distinct advantage in learning non-continuous and high variance mappings, since there is no constraint that the global function is produced by a summation of smooth functions. The SOM-based model will learn quickly when there are few data samples, as in the XOR problem, since the data points can be represented explicitly. In fact the XOR problem is ideally suited to the SOM-based approach partly because of the reasons above, but also because there is no requirement for generalisation of any kind. These intuitions are supported by the graph of figure 8.7 which indicates that the Kohonen-based approach outperforms the backpropagation system on this particular problem by at least an order of magnitude. Other features of a learning task that will recommend a local model include the requirement that a range of actions be accessible for any input stimulus, the expectation of temporarily discontinued regions of the input space, and also, possibly, a prior indication that the input distribution will be significantly non-equiprobable.

The distributed approach exemplified by the actor-critic backpropagation model confers its advantages in the form of superior generalisation and superior scaling potential. In problems where the number of input dimensions is high, or even where the dimensionality of the problem is low but where there is no correlation between the inputs, the backpropagation model would be expected to outperform any contender from the non-distributed model class. Support for this was seen in the task involving learning to throw projectiles under various parameterised conditions. The SOM-based application to a Khepera robot learning to avoid obstacles *was* successful in a six dimensional input space, but it is noted that in this instance there was considerable correlation between the inputs. In general, the curse of dimensionality can be expected to strike later in distributed learning models, which are therefore recommended for problems consisting of high dimensional input and output spaces, but equiprobable and stationary input distributions. Problems requiring a large amount of generalisation, and a smooth target mapping also suggest a distributed solution.

However, it is noted that the Q-AHC results reported in Rummery (1995) are disappointing and suggest that using backpropagation to represent dynamic, real-valued actions may be problematic. For example, in Rummery's particular robot navigation problem, it is reported that performance is no better, and sometimes worse, than using hand-coded actions. The problems considered in this chapter have been simpler than those considered by Rummery (although more complex than those considered by Gullapalli (1990)), and it is entirely possible that additional problems may arise from using backpropagation that have not been encountered in this chapter. Rummery considers an insufficient number of hidden units, local maxima in the action space, or function discontinuities as possible causes for poor performance, but no further investigation of these issues is offered. A more thorough comparison of local and distributed approaches to representation and generalisation in dynamic real-valued action RL problems ought to constitute future work.

8.6.1 Reliability

One final point is made regarding the reliability of the two model classes. The simplicity of the SOM-based model affords it a certain robustness in terms of training time, and resilience to local minima. A problem with standard backpropagation which has already been discussed is that it can be arbitrarily slow under certain conditions, where these conditions may depend on initial weight values, or on the data itself. There is also the question of whether or not either the actor or critic will get stuck in a local minimum. The sensitivity of the backpropagation algorithm to initial weight values is also well known (Bishop (1995),pg 260). Predicting these problems beforehand will itself be problematic, so in this sense the SOM-based model may provide a more reliable method if very little is known about the task, and there is no opportunity to gather empirical data.

CHAPTER 9

Discussion

This chapter first identifies some key issues pertaining to the applicability and scalability of the proposed model, and then concludes by considering some potentially fruitful avenues for future work.

9.1 Key issues to address

From the experiments and analyses performed in previous chapters, a number of key problems have emerged pertaining to the applicability and scalability of the proposed model. These problems are now summarised.

9.1.1 Design, diagnosis and analysis

It has already been noted how the Input map formation affects the reward estimation function which in turn influences how the Motor map forms. The Motor units control which actions are available and this alters the situations the agent finds itself in and hence the input to the Input map. The presence of such feedback loops makes design, testing and analysis difficult. For example, the design of the Input map may be functional and optimised until a change is made to the design of the Motor map, despite the fact that the latter comes after the former in the intuitive flow of control.

In general, striving for stable dynamics is found to be a considerable challenge, as it is often difficult to attribute a particular behavioural feature to a corresponding design feature. For example, a particular problem may seem to be in one or other of the maps, or in the reinforcement learning rules, or in the reward function, but may actually have its roots in a quite different part of the system. This represents something of an engineering nightmare because it is never possible to draw a line under the design and say "this works, now let's build the next bit". Instead, each part of the design has to be coaxed and cajoled into working in conjunction with the other parts. This also inflames the parameter problem since it is possible for parameters to interact in complex or unpredictable ways.

One avenue for addressing this problem may lie in building the system out of more flexible parts. For example, if each learning module could self-regulate its vital parameters according to the behaviour of the other modules, a significant burden could be unloaded from the designer. This idea would involve a shift in perspective with each part of the architecture (Input map, reward estimation, Output map) acting as an encapsulated module with the notion of 'environment' redefined in each case to include the other modules (as appropriate). This schema is consistent with the speculations of Minsky (1986), who hypothesises that the mind itself may be viewed as a society of specialised, encapsulated modules, with each achieving its local goals by either co-operating with or exploiting its neighbouring modules, without necessarily knowing anything about their internal workings. These ideas were discussed in section 3.3 as part of a brief review of the behaviour based model of intelligence. Greater flexibility and robustness may be the prizes on offer, but there is still a significant burden on the designer to construct and organise the parts in such a way as to achieve the emergence of the desired global behaviour.

9.1.2 Exploring the action space

Exploring the action space with unbiased random noise is a very basic approach which will become costly in high dimensional spaces. This observation is also made by Wedel and Polani (1996) who introduced the concept of *covariance learning* (section 4.5). Recall that their approach involved sampling the reward function at a number of points around a current action. These sampled reward values were used to approximate the reward function in the vicinity of this point, and then this reward model was used to estimate a useful direction for exploration (followed by a line search). Wedel and Polani (1996) used a Gaussian approximation, but the reward function could in principle be approximated by any function type, and any regression technique, including neural networks.

Other approaches to expediting the exploration of the action space could include using a-priori knowledge of the task to guide the search (see section 9.2.1), or maintaining some record of the expected return of various regions of this space, and exploring these regions with a probability proportional to this estimate so that regions that seem to be more fruitful are explored more. Clearly the best technique will depend on the kind of assumptions that can be made about the reward surface. For example, the last suggestion would seem to be most recommended when the reward function is close to stationary and varies only slowly with the action taken.

9.1.3 Finding high reward regions

Chapter 6 discussed the possibility that high regions of the reward surface may never be found by the proposed model if they are hidden in larger regions of lower reward. This was because action units were found to be attracted to large *areas* or *volumes*[1] of reward above $Mean(r)$. A standard hill-climbing algorithm is attracted to high *points* on the surface which is our goal here too, but the stochastic nature of the learning task precludes this kind of algorithm. For example, it is not possible to use the algorithm: "Sample a point on the reward surface and move straight to this point if the reward is greater than the reward at the current point". Instead we are obliged to maintain a

[1] These terms were used in the context of a two dimensional reward surface.

continuous learning rate which is annealed over time so that statistical tendencies are favoured over specific samples. Chapter 6 also suggested that modelling the reward surface, as used in Wedel and Polani (1996), may help ameliorate these problems, albeit at the cost of a new set of issues to address.[2]

9.1.4 Dynamic environments

We are also aware of potential difficulties associated with dynamic environments. In terms of mapping the input space and learning Q-values, maintaining small learning and exploration rates will always allow the system to adapt, eventually, to changes in the environment. But the situation in the Output space is quite different. Small learning and exploration rates in the Output map converge on a hill-climb, with an inevitable vulnerability to local minima. The implication is that if the reward function changes significantly, the plasticity in the Output map may have to be restored to its original and most plastic state. It is not currently clear how to determine when the environment requires this level of plasticity. See section D.5 for a discussion on parameters and the impact of dynamic environments.

9.1.5 Scalability

The key drawback of the proposed model is its scalability. As the intrinsic dimensionalities of the input and output spaces are increased, over-generalisation is inevitable — particularly once the dimensionality of the underlying maps are exceeded. It was also seen in section 8.5.2 how problems can arise if only one dimension out of many other noisy dimensions is relevant for classifying the input space. These problems were put down to the 'local' representation employed by the SOM, as well as the equality of influence automatically given to each element of the input vector.

[2] The idea of the *reward surface*, here and in chapter 6, would be replaced by the *return surface* in the general case of delayed rewards.

9.2 Future work

Some possibilities for varying this work have already been discussed in chapter 6, such as using different representations for the input space, using bootstrapping value estimation techniques, and investigating a number of alternative learning and exploration rules for the Output map. There are clearly also some challenges in addressing the issues raised in the previous section. Apart from these ideas, a number of other avenues for future work are now suggested.

9.2.1 Biasing learning

Reinforcement learning is notorious for its slow learning, and the discussion of the Output map parameters in section D.3 suggests that this particular part of the system might be even slower. This is partly because it depends on the Q-learning for guidance, partly because it must make many samples before being able to make an accurate statistical judgement as to where to explore next in the action space, and partly because the exploration is random. It seems certain that in a large dimensional space, with a complex reward function and a large number of actions to learn, the algorithm presented here will be prohibitively slow and discover only suboptimal sets of actions.

One way to address this problem is to consider biasing the learning. So far, the Output map is given no a-priori information and is working from scratch. A possibility is to bias the learning by starting the units in approximately the right positions (assuming this is known) and then letting the algorithm do the fine-tuning. This is perhaps analogous to calibration learning, and allows for adaptation to sensory-motor drift as well as internal or external damage. The potential for spontaneous development of new actions would not be precluded, but the size of the learning problem could be reduced by removing the need to autonomously discover every action from scratch.

Another way to bias learning would be to provide guidance in the form of regular updates to the Output map in the direction of innately specified actions, rather than always in the direction of spontaneously generated actions. For example, if it is known that some subset of a collection of actions is likely to be useful to the agent, but not specifically which ones, then the system could be guided by fixed and periodic updates of the map towards each of the potentially useful actions. The ones that actually did turn out to be useful would be reinforced and maintained.

It seems likely that biasing the formation of the Output map in some way would be an important part of any serious application of this system. Of course the Input map and the Q-values may be biased in a similar way, although the emphasis is on the formation of the Output map because this is considered to be the bottleneck of the system. Kaelbling et al. (1996) identifies the specification of innate knowledge in one form or another as a fundamental technique for scaling RL to non-toy problems.

9.2.2 Dynamic resolution of the Input map

One key area of interest is mapping the input space in a manner more specialised to the task in hand. Non-equiprobable firing rates and unfaithful distribution of units may not turn out to be too serious a problem, but a core concern of any state representation technique is to produce categories which are as consistent as possible with respect to how the reward signal behaves under various actions. Some regions of input space may require a higher resolution of representation than others to achieve this, irrespective of the relative frequency of the distribution within these regions. In the proposed model, the density of units crudely reflects the frequency of stimuli, but no attempt is made to map the space with respect to the reward signal. One approach may be to alter the learning rule of the Input map so that there is a bias towards units that are poor predictors of reward and hence to regions requiring finer grained representation. One way to achieve this might be to set the learning rate and neighbourhood of the Input map proportional to the error in the Q-value of the currently active Input unit. In this way, if an Input unit is a poor predictor of its own reward, it will tend to pull its neighbours in with more force, thus increasing the resolution of the map in its vicinity.

9.2.3 Delayed rewards

The longest reward horizon used in the experiments of this thesis was $h = 4$ in the task of the Khepera robot learning to avoid obstacles. Although the issue of delayed rewards is not of immediate interest to this thesis, the proposed model should at least be tested on a problem where returns have potentially long horizons. While delayed rewards are known to be no theoretical problem for Q-learning itself, here we are specifically interested in the implications for learning in the action space.

An experiment is outlined to test the robustness of the proposed model to delayed rewards: The cart-pole balancing problem (see section 4.5 for a brief description) is a relatively simple task that requires a continuous range of actions, with each action interpreted as a force applied to the cart to keep the pole upright. It is true that the pole may be kept upright by interleaving discrete actions, but the issue of continuous actions can be made more salient by introducing latency into the mechanics, by restricting the number of actions that can be made in a fixed time, or just by emphasising the importance of smoothness, efficiency and robustness. If the reward signal only comprises feedback associated with the pole dropping, then the problem is potentially one with very delayed reward, and provides an interesting context within which to explore the problem of learning continuous actions.

Clearly some minor changes would first need to be made to the algorithm. In particular, authentic Q-learning, or a similar bootstrapping value estimation technique such as SARSA would be recommended, since these automatically deal with arbitrarily delayed reward. This involves changing the criteria in steps ❻ and ❼ (page 124) for updating Output units from:

❻ Calculate the discounted reward of the state-action pair of h time-steps ago now that all the reward is in for that pair:

$$R = r_{t-h} + \gamma r_{t-h+1} + \gamma^2 r_{t-h+2} + \ldots + \gamma^h r_t$$

❼ If $R > Q(U_{t-h}, A_{t-h})$ then update the Motor map towards $\langle M_L, M_R \rangle$ of h time-steps ago, proportionally to the Motor neighbourhood function, ψ_1, and Motor map learning rate, OL.

to:

❻ Estimate the full discounted reward of the the state action pair, (U_t, A_t):

$$R = r_t + \gamma \max_{A_{t+1}} Q(U_{t+1}, A_{t+1})$$

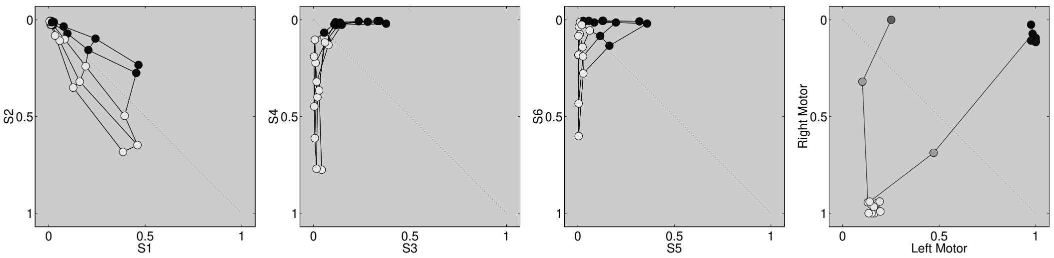
❼ If $R > Q(U_t, A_t)$ then update the Motor map towards $\langle M_L, M_R \rangle$ proportionally to the Motor neighbourhood function, ψ_1, and Motor map learning rate, OL.

Apart from the need to test robustness to delayed rewards, using a bootstrapping technique such as authentic Q-learning in the reward estimation module would also make the proposed model more consistent with current practice. In principle, *any* value estimation technique could by used here, providing of course it suits the task in hand.

9.2.4 Abstract categories

Recall the original experiment involving a Khepera robot learning to avoid obstacles. The final positions of the Input and Output maps are duplicated in figure 9.1. By looking at the 25 Input units, and helped by our intuitions of the problem, we were able to propose that the system had dichotomised the input space, with one half corresponding to situations involving more left sensory activation than right, and the other half to situations involving more right sensory activation than left. Furthermore, by looking at the 20 Output units, we observed two distinct actions — a hard left turn on the spot and a hard right turn on the spot — with the appropriate action being taken consistently within each of the two regions of the six dimensional input space.

Paraphrasing the solution for the obstacle avoidance problem in this way involves generating symbols and then using these symbols to construct rules. In this case the four symbols are:



(a) (b) (c) (d)

Figure 9.1: The Input (a,b,c) and Motor maps (d) after learning. Motor units are coloured according to their position in the action space (see figure 5.20) and Input units are coded according to the Motor unit for which they have the highest Q-value. Duplicated from figure 5.19.

$S_1 =$ Region of state space with more left sensory activity than right
$S_2 =$ Region of state space with more right sensory activity than left
$A_1 =$ Left turn
$A_2 =$ Right turn

and the rule is:

$$\text{If } S_1 \text{ then } A_2 \text{ else } A_1$$

Symbols are clearly powerful tools for both representation and communication providing an appropriate set can be found. In many cases the designer may not have access to such symbols a-priori, but in the example described here the symbols are actually implicit in the agent's own low level representations that it gained through interaction with the problem. One avenue for extending the current model would be to search for a technique to abstract out these higher level symbols from the low level state-action-reward information acquired during learning. This could potentially lead to *grounded symbols* which could then be ploughed back into more productive rounds of environmental interaction, especially those involving related tasks. For example, if the problem above could be reduced to just two state units (one for each distinct half of the state space) and two action units (one for each type of turn), then the number of Q-values in the system is reduced to only four.

The psychologist and cognitive scientist, Karmiloff-Smith, discusses child development in her book, *Beyond Modularity* (Karmiloff-Smith, 1995), with an emphasis on the iterative, incremental and interactive aspect to forming representations of the world. Her thesis could be paraphrased as advocating a process of abstraction (or in her more general words *redescription*) which takes place on a child's representations of a problem. The new representations that are formed as a result of this redescription can then be used to fuel a new round of environmental interaction that can be both more efficient and more effective. This new round of interaction can then generate more salient and appropriate symbols and so on. She provides evidence for the redescription occurring in stages with development a step-wise process and the transition from one set of representations to the next only occurring when sufficient confidence is held in the status of the new representations. This supports the hypothesis that these representational changes are qualitative, and the product of something more than just a slow, continuous and homogeneous adaptive process.

With these thoughts in mind, appendix F performs some preliminary investigations into abstracting over the standard representations generated by the proposed model. Some simple experiments are performed which demonstrate such a process in action and some possible implications for such a process are also discussed. As an example of the benefits of abstraction, consider that the problem of delayed rewards is caused by many states and actions intervening between an original state-action pair and its reward. If a mouse had to learn to navigate a maze for a food reward by fusing low level sensory data with low level motor commands, its task would surely be hopeless because there would be too much data to organise. In any case, there is no need to reinforce the low level actions of putting one foot in front of the other, and no point in trying to associate hundreds of low level perceptual cues with these actions. Instead, in the context of this task, the mouse probably need only consider each junction of the maze as a state, and associate it with a higher level action of turning in one direction or another. Now the mouse's history parameter, h, need only be a few actions deep, and the issue of delayed rewards is ameliorated.

9.2.5 Local reward

With respect to addressing the issue of delayed rewards, the alternative to reducing the number of state-action pairs intervening between a particular pair and its goal, is to concentrate on making the reward signal as local as possible. With respect to intelligent robots, both Kaelbling et al. (1996) and Mataric (1997) cite this as a key scaling issue, and as we saw back in section 2.9.2, the progress estimators used by Mataric (1994, 1997) go some way to addressing this issue. In the work of this thesis, local reward signals have been used throughout to avoid problems with delayed reward, since delayed rewards have not been the focus of the thesis. However, apart from the cart-pole experiment proposed in section 9.2.3 to verify the proposed model under delayed reward conditions, further general research needs to be invested in the *autonomous* derivation of a local reward signal from infrequent goal states. One simple example would be to estimate reward locally according to some appropriate measure of distance between the current state and known goal states.

9.2.6 Combining backpropagation and the SOM

In chapter 8 the relative merits and limitations of using backpropagation and the SOM to generalise over the state and action spaces were compared. The major drawback of using a SOM to generalise over the state space lay in its potential lack of scalability. Particular issues were input spaces with a high intrinsic dimensionality, and the problems caused by each input line to the SOM having equal weight in the classification process. However, maintaining a SOM in the Output space was deemed to be advantageous because the generalisation was dynamic (unlike Lin's QCON model), allowed multiple actions to be maintained for each state (unlike Gullapalli's SRV units), allowed maximum reuse of learned actions across states (unlike the Motoric map), embraced the central notion of explicitly estimating expected reward (unlike the CRBP algorithm and Touzet's backpropagation approach), and made searching for the optimal action for a given state straightforward (unlike some coarse coding techniques)[3]. A specific problem identified with using backpropagation to generalise over the action space was caused by non-stationary environments (recall section 8.5.3) in which the network could be lured into over generalisation.

[3] See section 6.7 and chapter 8 for more detailed comparisons.

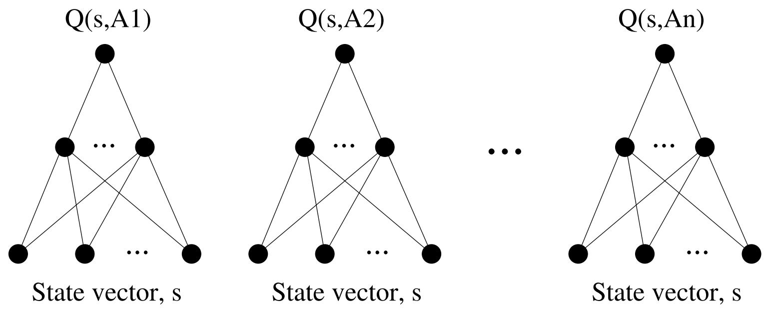


Figure 9.2: Lin's *QCON* model where a separate backpropagation network for each action, $\{A_1 \ldots A_n\}$, learns to map states to Q-values under those actions. Duplicated from figure 4.4.

With these arguments in mind, the discussion returns to the QCON model of Lin (1993) which is summarised again in figure 9.2. In the comparison of section 6.7, Lin's model was favoured, with the exception of the undesirable feature that the actions were fixed. An obvious extension to the proposed model, inspired by Lin's QCON, is shown in figure 9.3. The idea is that the Output map is formed in the usual way, except that a separate backpropagation network is used for each action to map states to the Q-values of those states under that action. The intention is to combine the powerful and flexible generalisation power of backpropagation in the state space, but to retain the desirable properties of the SOM in the output space. In our opinion this kind of hybrid model deserves further investigation.

9.2.7 The auto-associative MLP

Continuing the search for a more sophisticated and scalable way to generalise over the state space, the auto-associative multi-layer perceptron is identified as another candidate (Rumelhart et al., 1986). Figure 9.4 shows the basic idea with the network being trained to map each n-dimensional input vector to itself. Because the hidden layer contains fewer units than the input and output layers, the data is compressed in the activations of the hidden layer. However, since there is only a single layer of weights between the input and hidden units, the compression is linear and in fact this model has been shown to be equivalent to principal component analysis, with the m hidden units corresponding to the first m principal components (see Bourlard and Kamp (1988); Baldi and Hornik (1989), reported in Bishop (1995)). This equivalence depends on

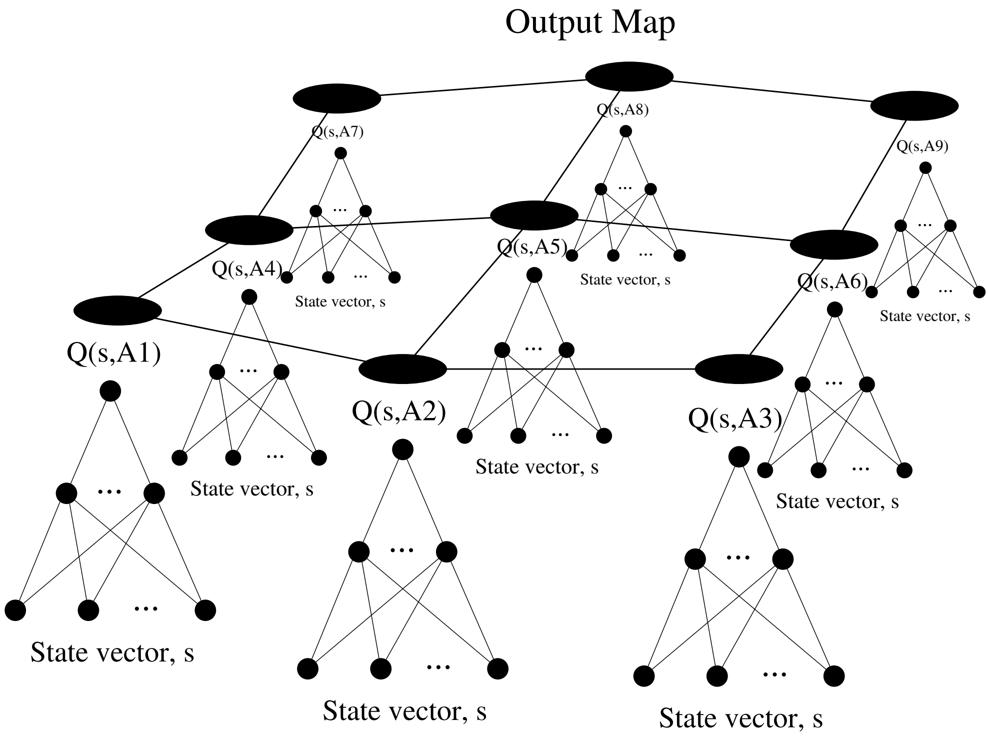


Figure 9.3: An adapted architecture inspired by Lin's *QCON* model. As in QCON, a backpropagation network is used to provide powerful and flexible generalisation over the state space. As in the proposed model of this thesis, the actions are dynamically generated using a SOM in the action space.

the hidden units being linear and the network's weights minimising a sum-of-squares Error function.

Even if non-linear units are used in the hidden layer, the network of figure 9.4 is still restricted to a linear compression of the input space. In other words the input data is projected onto an m-dimensional hyperplane within the original input space. However, by introducing *two* layers of weights between the input and hidden layers, the model can be adapted to perform non-linear principal component analysis, as in figure 9.5 (see Bishop (1995), pg 316). The mapping between the input and hidden layer can now be completely arbitrary, at least in principal (see page 48), and so the input space is effectively projected onto a *non-planar* m-dimensional subspace of the original input space. However, finding a suitable set of weights becomes a potentially costly business, particularly given that there are now four layers of weights to which to backpropagate error.

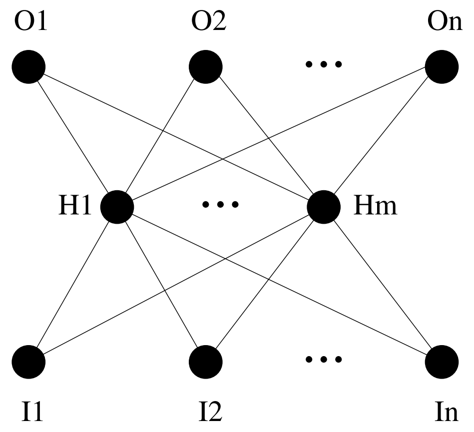


Figure 9.4: An auto-associative MLP which compresses the data into the lower dimensional representations of the hidden layer. The approach is shown to be equivalent to principal component analysis.

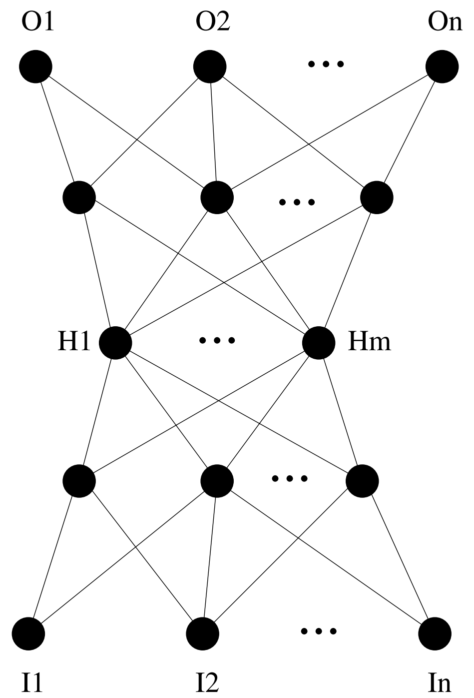


Figure 9.5: The hidden layer can now represent a non-linear compression of the data, because of the two layers of weights between the input and hidden layer. This results in a non-linear principal component analysis.

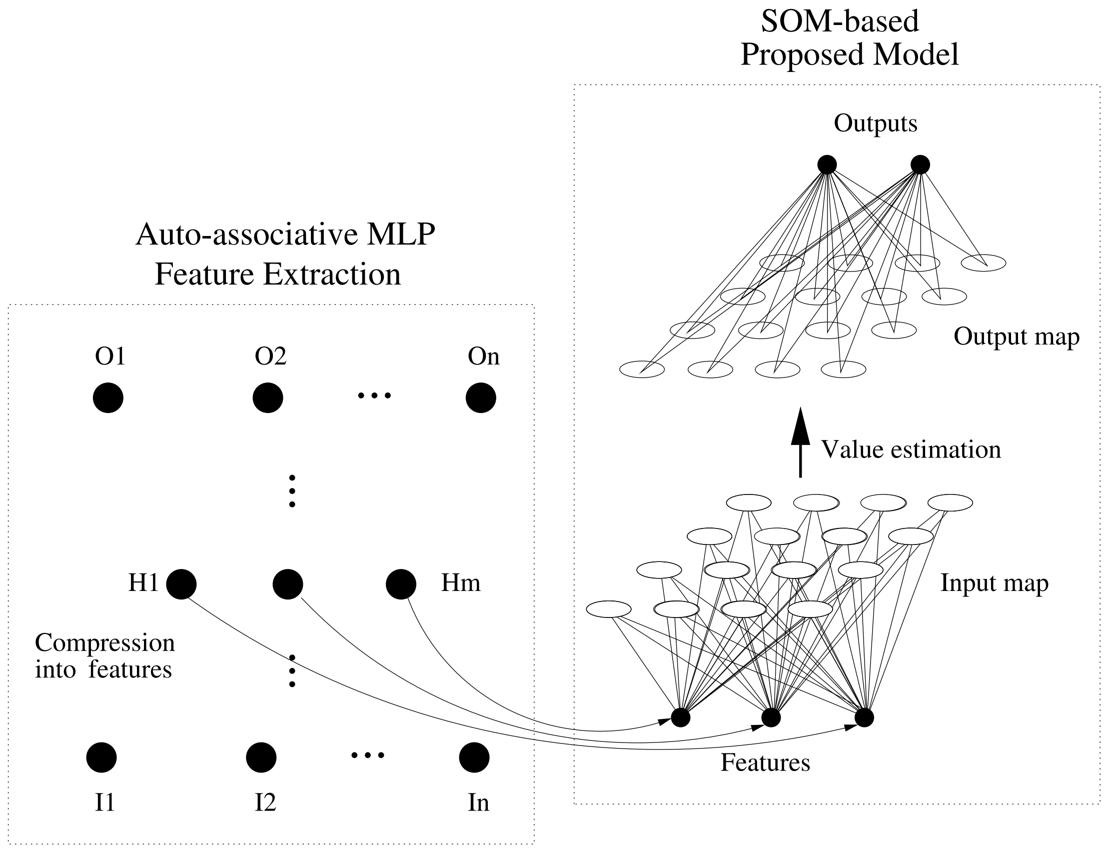


Figure 9.6: Using the features extracted by an auto-associative MLP to represent the input data to the proposed SOM-based model. Now many of the drawbacks identified with using a SOM to map the input space are addressed.

One obvious way to make use of the dimensionality reduction offered by this technique is to use the activations of the hidden layer as inputs to the Input map of the proposed model (figure 9.6). In this way, non-linear feature extraction is performed as a kind of dynamic dimensionality reduction preprocessing step. This approach could potentially ameliorate some of the drawbacks of using the SOM for mapping high dimensional input spaces.

However, the usual problems of local minima and long training times apply, and this may be aggravated by the fact that the input distribution is unlikely to be stationary. The number of units in the hidden layer also needs to be tuned by hand. Another issue is that the feature extraction is effectively performed without respect to the task in hand. In other words the principal components of the data may not actually be the relevant features for categorising the input space (see Bishop (1995), page 318, for an example). To address this, there may be other approaches worth investigating including training the MLP not to its own inputs, but to the reward signal for example, so that the structure identified in the input space bears more significance to the learning task in hand. There are undoubtedly other ways in which to incorporate the powerful generalisation capabilities of the MLP into the type of RL problem considered here without sacrificing the desirable properties identified in figure 6.25.

Combining neural networks is by no means a new idea. In fact Tani and Fukumura (1994) also combine an MLP and a SOM in a robot learning task, but their approach is to first use the SOM, and then use the index of the winning SOM unit as input to a backpropagation network which goes on to perform a classification task. Their idea is that the SOM reduces the dimensionality of the data for the benefit of the backpropagation learning process.

9.2.8 Combining RL and the behaviour based model

Apart from using local reward signals, abstract state and action symbols, and biasing learning, a key technique for scaling RL is the application of the behaviour based model. We have already seen examples of how breaking a problem up into smaller behaviours can facilitate learning within an RL domain (see section 5.6.5, Mahadevan and Connell (1991), Dorigo and Colombetti (1994), and Dorigo (1995), for example), and an important thrust of RL research must be directed at achieving this in a less ad-hoc, and more principled and autonomous way. It is noted that while both RL and the behaviour based approach have independently yielded ground-breaking applications, the two in combination are yet to achieve similar accolade. Getting the most out of combining RL and BB is clearly an important area for future work.

Kolmogorov's theorem[4] (Kolmogorov, 1957) guarantees that any continuous multivariate function can be represented exactly with a finite number of functions of a single variable composed only by addition. In terms of RL, splitting the target function up

[4] Accounts of this theorem are found in Bishop (1995), pg 137, and Rojas (1996), pg 265.

in this way could be useful because generalisation would only need to be considered in low dimensional state and action spaces, which in turn would make the SOM approach to representation both appropriate and scalable. Kolmogorov's theorem does not provide a method for achieving this decomposition, and so the challenge lies in the construction and coordination of an appropriate set of smaller functions. This clearly overlaps with the aspirations of the behaviour based community for whom a method to autonomously construct and compose behaviours to produce a desired global behaviour is something of a Holy Grail.

One final point about the behaviour based model and learning real-valued actions is made. The motivation for learning real-valued actions appealed to those applications where interleaving discrete actions was deemed inappropriate. However, achieving a desired global behaviour out of interleaving behaviours is precisely the founding principle of the behaviour based methodology. The conclusion is that while the proposed model is shown to work well within a behaviour based framework, given that BB is appropriate then adaptive real-valued actions may not actually be necessary. More work is required to explore the possibility of mutually supportive niches for continuous RL and the behaviour based paradigm.

9.2.9 Rigorous analysis

Finally, notwithstanding chapter 6, this thesis requires a more rigorous analysis. The exact dynamics of the formation of the Output map in the full system are unclear, and therefore so are the limitations of the proposed model. In particular, further research needs to be carried out to determine the implications of the criteria for deciding when and how to move Output units in the output space. Although empirically found to be effective, learning towards randomly perturbed actions if the sampled reward appears greater than the current estimate requires additional analysis before any claims regarding convergence and efficiency can be made. We would also like to know how self-organisation in the Output map will affect these claims.

CHAPTER 10

Conclusion

This chapter restates the goal of the thesis, reviews the motivation for trying to achieve this goal, and then summarises the main results and achievements.

10.1 The problem

In chapter 1, the standard reinforcement learning problem was initially summarised by figure 10.1, in which the aim was identified as achieving an optimal mapping between a set of discrete states and a set of discrete actions. We defined 'optimal' as maximising the expected reward over either a finite or infinite horizon, in accordance with the standard RL theory reviewed in chapter 2.

The need for generalisation was then discussed in the context of large or continuous state and action spaces. This led to the more general problem of figure 10.2 and a declaration of the focus of the thesis as:

> *"The learning of optimal mappings between a continuous input space and a continuous output space, of arbitrary dimensions, in order to maximise a reward function that yields a scalar value for each sampled state-action pairing."* (page 6)

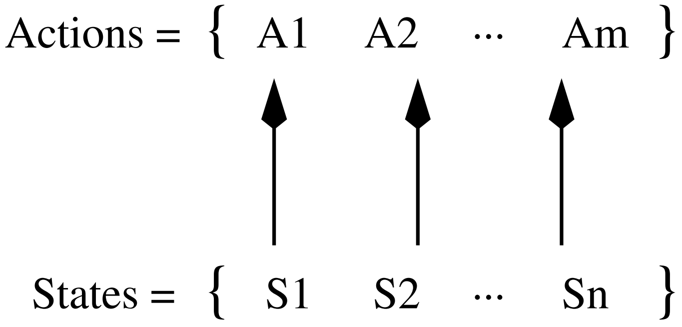


Figure 10.1: Summary of the basic reinforcement learning problem. Duplicated from figure 1.1.

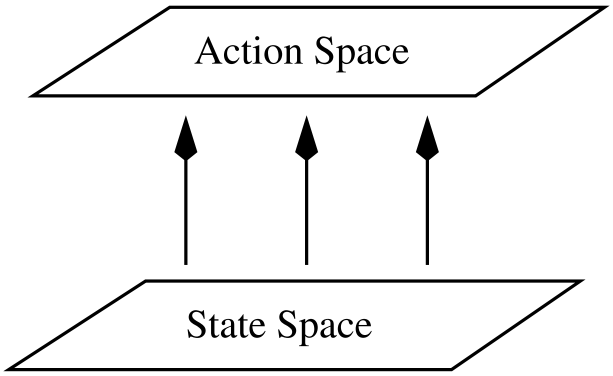


Figure 10.2: Summary of the more general reinforcement learning problem involving continuous state and action spaces. Duplicated from figure 1.2.

10.2 Motivation

Chapter 7 introduced the problem of learning to throw a projectile to a target under parameterised conditions which necessitated the online adaption of real-valued actions. However, it was noted in chapter 1 that an optimal solution to many learning tasks could be approximated by suitably interleaving a number of discrete actions. One example offered was approximating a continuous robot trajectory by frequently switching between a forward behaviour and left and right turns. The idea is analogous to controlling the continuous temperature of a fridge with two discrete actions corresponding to the thermostat being on or off. This led to the proposal that in many cases it may be sufficient to simply hardwire a small number of discrete actions as in the work of Mahadevan and Connell (1991) for example.

However, a number of circumstances requiring or at least recommending the dynamic generation of real-valued actions were discussed. One such circumstance was latency in the system such as one might find in steering a boat or guiding a missile for example. In these cases, a large momentum to maximum-correcting-force ratio suggests that actions need to be as accurate as possible. This idea was taken to the extreme in the projectile throwing task because here there was no corrective capability whatsoever.

It was also noted that using a series of oscillatory movements to approximate a smooth, continuous movement has potential negative implications for time-critical applications. As an example, we considered the success of a hypothetical squash or table-tennis player who uses oscillating discrete actions to approximate trajectories for moving and intercepting the ball. Any problem in which the number of actions possible in a fixed time is limited will generate a timeliness issue.

We also realised that even if a small set of discrete actions did happen to exist which could be used to approximate a desired set of continuous actions, such a set may not actually be available to the designer a-priori. Additionally, adaptable real-valued actions may enable a system to adapt efficiently to small environmental changes, perhaps as a result of sensory-motor drift or some kind of internal or external damage.

The motivation for the thesis was then stated as:

> *"While abstract problems such as games can be played using an a-priori defined set of discrete actions, many real-world problems require an adaptable range of continuous actions. Even where discrete approximations are possible, issues of speed, efficiency, latency, and reliability may recommend the use of adaptable real-valued actions."* (page 8)

10.3 Desirable properties

A review of the literature in chapters 2 and 4 highlighted a number of existing approaches to the generalisation of both the input and output spaces. Of particular interest were those concerned with dynamic generalisation over the action space which included the Motoric map of Ritter et al. (1990) with the covariance extension of Wedel and Polani (1996), the CRBP algorithm of Ackley and Littman (1990) and its derivatives (Ziemke, 1996), the QCON model of Lin (1993), two approaches in Touzet (1997) based on the SOM and backpropagation, and the SRV unit of Gullapalli (1990). A number of key properties were identified as being valuable to any candidate model, but it was noted that none of the existing models appeared to satisfy all these criteria.

Desirable properties:

- Dynamic generalisation.
- Close adherence to RL theory with estimating expected reward at the centre of a system whose aim it is to maximise this expectation.
- Support for a real-valued, discounted reward signal.
- Support for real-valued states and actions.
- Maximum re-use of actions across different states.
- The maintenance of multiple actions supporting a choice of outputs in each state.
- Simplicity and efficiency.
- Scalability.

10.4 The proposed model

Chapter 5 then proposed a new model based on two Kohonen maps — one in the input space (Input map) and one in the action space (Output map) (see figure 10.3). The Input and Output maps are responsible for condensing the two spaces into a small set of discrete states and actions (respectively) so that a standard value estimation technique such as Q-learning can be utilised. In this way, each unit of the Input map becomes a state of the problem, each Output unit becomes an action, and for every state-action pair a ‘Q-value’ is maintained.

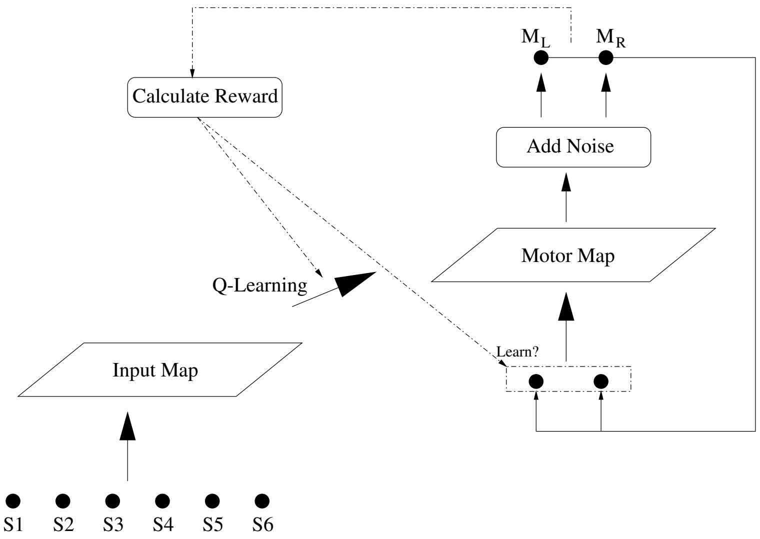


Figure 10.3: Basic architecture. Duplicated from figure 5.16.

While the formation of the Input map is passive, the Output map has to explore the action space as well as generalise over it. This is achieved by using random noise to perturb each proposed action, and then by learning towards that perturbed action if it appears to be an improvement on the original proposed action. The topology preservation property of the SOM also provides a means to speed up the propagation of reward information around the system. Q-values are able to learn from their neighbours in the Q-table according to the neighbourhood functions of the two maps, as in figure 10.4.

The proposed model was shown to perform well on a number of different problems including training a simulated Khepera robot to avoid obstacles. This involved generating a mapping from a continuous six-dimensional sensory state space to a continuous two-dimensional motor space. The model was also shown to be robust to internal damage, and to perform well within a behaviour based architecture, as well as on a number of non-discounted regression problems, including one requiring a non-contiguous mapping between state and action spaces. The proposed model was also seen to satisfy many of the desirable properties identified earlier, and was deemed to compare favourably with the other reviewed techniques for generalising over the action space (see the comparison table of figure 10.5).

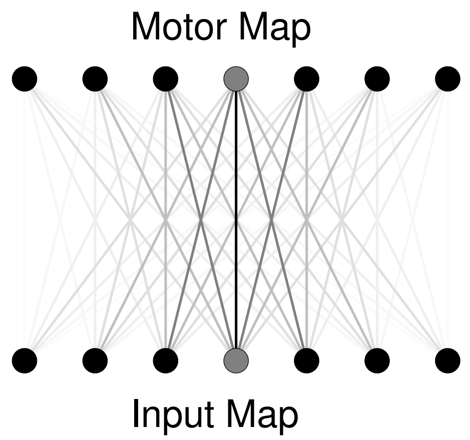


Figure 10.4: Whenever a Q-value is updated, all other Q-values are also updated proportionally to the product of the neighbourhoods of the original Input and Motor units. Duplicated from figure 5.15.

According to the comparison table, the simplest non-neural approaches score well, but lack adaptability and scalability. The existing SOM-based approaches exhibit inflexibility, and most of the backpropagation approaches are forced to make significant departures from the underlying theory in order to accommodate an ostensibly supervised technique within an RL framework. Notable exceptions are the work of Gullapalli (1990) and Rummery (1995) which, along with Lin's QCON algorithm are considered good foundations upon which to build future work. The inflexibility in both action exploration and action selection emerges as a consistent weakness of the backpropagation-based approaches, and there is also some doubt over their scalability and efficiency. These last two points pertain to potential problems with local minima, long training times, and other issues relating to interpretation, diagnosis and robustness to non-stationary environments.

The proposed model achieves most of the objective criteria, although there are some key concerns over scalability. These concerns relate to the local representation of the SOM, the requirement that the data has a low intrinsic dimensionality, and the fact that each input line contributes equally to the winner selection metric. An example of a problem caused by the latter occurs when only one of many input lines is relevant for making a discriminatory decision.

In chapter 8 these problems were investigated further through a direct comparison

		Coarse-coding					SOM			Backpropagation						
		Handcoding (Mah. & Con.)	Coarse-coding	RBF (Santamaria)	CMAC (Prescott)	Nearest Neighbour (Moore)	SOM (Touzet)	Motoric map (Ritter et al.)	Covariance learning (Wed. & Pol.)	CRBP (Ackley et al.)	CRBP (Ziemke)	QCON (Lin)	Backprop (Touzet)	SRV units (Gullapalli)	Q-AHC (Rummery)	PROPOSED MODEL
	Dynamic generalisation	✗	?	?	✓	?	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓
Generality	Adhere to RL theory	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✓	✗	✓	✓	✓
Generality	Real-valued, discounted reward	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✓	✗	✓	✓	✓
Generality	Real-valued states and actions	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
Flexibility	Re-use of actions	✓	?	?	✗	?	?	✗	✗	✗	✗	✓	✗	✗	✗	✓
Flexibility	Multiple actions for each state	✓	✓	✓	✗	✓	✓	✗	✗	✗	✗	✓	✗	✗	✓	✓
Flexibility	Interpolation	✓	✓	✓	✗	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗	✓
Scalability	Low update cost	✓	?	?	?	✓	✓	✓	?	✓	✓	✓	✓	✓	✓	✓
Scalability	Low access cost	✓	✗	✗	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
Scalability	Low memory use	?	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✗
Scalability	Scalable	?	?	?	?	?	✗	?	?	✗	✗	?	✗	?	?	?

Figure 10.5: A comparison of the proposed model with the reviewed approaches to action space generalisation. Duplicated from figure 6.25.

with a backpropagation model based on both the early actor-critic model of Barto et al. (1983) and the more recent SRV unit of Gullapalli (1990). Comparisons of backpropagation with the SOM for generalising over state and action spaces suggested two distinct model classes — those utilising *local* representation (the SOM) and those utilising *distributed* representation (backpropagation). In the case of distributed representations, every output depends on every free parameter for every input, and every free parameter is updated on every learning cycle, whereas in the local representation model, parameters have a more explicit scope. Although backpropagation was judged to benefit from maintaining distributed representations in terms of robustness to high dimensional data spaces and an ability to discriminate between the importance of different input lines, a number of key disadvantages were identified too. In particular, the experiment of section 8.5.3, in which half of the input distribution was discontinued and the reward function over the other half changed, revealed that the proposed backpropagation model was vulnerable to over-generalisation in non-stationary and non-equiprobable environments. Sensitivity to the order and frequency of presentation of the learning data was further noted in the XOR problem (see section 8.4.1).

These and the more standard issues of local minima, long training times, the difficulty of generating choices of actions, and difficulties in interpretation and diagnosis, raised some doubts as to the suitability of backpropagation for the kind of interactive learning problems of interest to this thesis. These doubts were then weighed up in comparison with the scalability drawbacks of the SOM-based model leading to a number of suggestions regarding the kind of applications each model class may be best suited to.

10.4.1 Ideal applications

In particular, the experiments suggested that the SOM-based model will have a distinct advantage when learning non-continuous, high variance mappings (such as the XOR problem, for example), since there is no constraint that the global function is produced by a summation of smooth functions. The SOM-based model will learn quickly when there are few data samples such as in the XOR problem since the data points can be represented explicitly. In fact the XOR problem is ideally suited to this approach partly because of the reasons above, but also because there is no requirement for generalisation of any kind. These intuitions were supported by the graph of figure 8.7 which indicated that the Kohonen-based approach out performs the backpropagation system on this particular problem by at least an order of magnitude. Other features of a learning task that will recommend a local representation model include the requirement that a range of actions be accessible for any input stimulus, the expectation of temporarily discontinued regions of the input space, and also, possibly, a prior indication that the input distribution will be significantly non-equiprobable.

The distributed approach exemplified by the actor-critic backpropagation model was seen to confer its advantages in the form of superior generalisation and superior scaling potential in terms of both robustness to high dimensional data spaces and an ability to discriminate between the importance of different input lines. In general, the curse of dimensionality can be expected to strike later in distributed learning models, which are therefore recommended for problems consisting of high dimensional input and output spaces, but equiprobable and consistent input distributions. Problems requiring sophisticated generalisation, and a smooth target mapping also suggest a distributed solution.

A final point was also made about the respective reliability of the two models. In general it was concluded that given the potential failings of the two systems, the SOM-based model is probably more robust, being less sensitive to learning rates, initial conditions, and peculiarities of the input distribution, thus providing a more reliable method if very little is known about the task.

10.5 Issues to address

Apart from the general pros and cons of the SOM-based model class compared with more distributed approaches, the analysis of chapter 6 uncovered some problems specific to the proposed model that remain unresolved by this thesis. These issues were summarised in the discussion of chapter 9 as:

- Dependencies amongst Input map, Output map and value estimation modules make design, diagnosis and analysis problematic.
- The use of random noise to explore the action space is potentially slow.
- The Output map is not guaranteed to discover regions of highest reward on the return surface. Because Output units are attracted to large areas or volumes above $Mean(r)$, at least in the early stages of plasticity, optimal regions may effectively be hidden within larger regions of lower reward. This is deemed to be a side-effect of using a *stochastic* hill-climb.
- While the Input map and value estimation module may be robust to dynamic

environments (by maintaining small residual learning rates), the Output map may be susceptible to local maxima. As plasticity in the Output map is annealed, a standard hill-climb of Output units is performed over the return surface. If the reward function has changed significantly since the annealing of plasticity in the Output map, the chance of suboptimal behaviour is increased.

- Scalability restrictions.

10.6 Future work

The discussion of chapter 9 also suggested some potentially fruitful avenues for addressing these problems. Of particular interest are hybrid models which aim to combine the advantages of the SOM and backpropagation. A number of potentially interesting architectures were proposed which use backpropagation to generalise over the input space, but maintain a Kohonen map in the Output space. Although these hybrid models were not tested, the hope is that they could satisfy all of the desirable criteria in figure 10.5.

Other proposed future work includes biasing learning, constructing abstract categories and producing grounded symbols (see appendix F), and altering the update rule of the Input map so that the resolution of the map reflects the variation in the reward function as well as the relative density of the input data.

10.7 Conclusion

This thesis concludes that the self-organising map can be used in conjunction with current RL theory to provide real-time dynamic representation and generalisation of continuous action spaces. The proposed model is shown to be efficient and robust and judged to be unique in addressing and satisfying a number of desirable properties identified as important to a large class of RL problems.

Appendix A

The SOM neighbourhood function

The neighbourhood function preferentially weights the influence of a unit on its nearest neighbours in a manner indicated in figure A.1.

The exact way in which the neighbourhood function, ψ, was calculated was not found

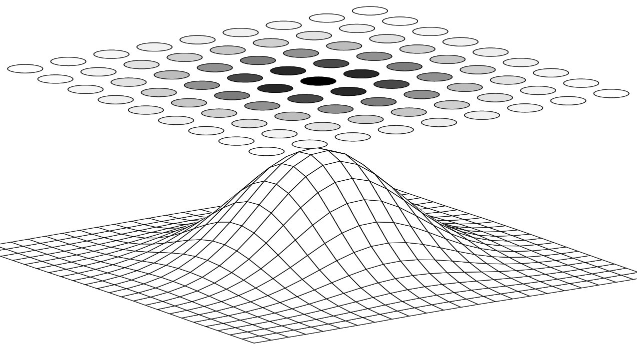


Figure A.1: A common neighbourhood function in which the influence of the winning unit on near neighbours is greater than those at a distance. The strength of shading of the units in the map reflects the height of the neighbourhood function underneath, and corresponds to the value of the term $\psi(winner,t)$ at that point (see section 3.2.2). In this case, the winning unit happens to be in the centre of the map.

to affect the formation of the map. However to clarify the Kohonen update rule, the calculation of ψ is made explicit: For each unit in the map, each element of that unit's weight vector is moved towards the corresponding element of the input vector proportional to $IL \times \psi(IN, d)$ where IL is the learning rate of the map, IN is the neighbourhood size parameter, and d is the distance in units between the unit being updated and the winner for this stimulus. $\psi(x, y) = z^y$ where $z^x = 0.05$. This equation implements a neighbourhood function similar to that of figure A.1, with 0.05 corresponding to the height of the surface at a unit that is distance IN away from the winner. In this way, as the neighbourhood size parameter, IN, is reduced, the peak of this surface decays quicker and quicker until when the neighbourhood is 0, ψ yields a value of 1 for the winning unit and less than 0.05 for all other units. For practicality, units yielding $\psi < 0.05$ are ignored. The value of 0.05 above is chosen arbitrarily.

Clearly other neighbourhood functions are feasible (a Gaussian function is a popular choice in the literature), and it is noted that the formation of the SOM is not particularly sensitive to such matters providing the size of the neighbourhood is roughly appropriate for the current stage of the map. In the experiments performed as part of this thesis, the neighbourhood was typically initialised to a large proportion of the map, and annealed either to a single unit or to a single unit and its immediate neighbours.

The neighbourhood function was not formally investigated as part of this thesis beyond a few empirical trials. The reader is referred to standard texts on the SOM (Kohonen, 1987, 1995) for an analysis.

Appendix B

Peripheral Design Issues

There are some algorithmic details that are not considered essential, but are included here for completeness. During the development of the algorithm a large number of techniques and parameters were experimented with — too many to perform a principled analysis of each one, particularly given the interactions and interference that was often observed between them. However, a few techniques emerged as being generally favourable, although their exact influence is not measured. The only justification for such an unprincipled approach is that I do not consider these points critical to the operation of the algorithm, and mention them not because they are guaranteed to have a significant or even positive impact on results, but rather because they are representative of the kind of issues that came under consideration during the development of the system. A formal analysis of every dimension of variability is simply outwith the scope of this thesis.

The first detail is that when an action unit explores to produce the perturbed action, $\langle M_L, M_R \rangle$, if this point lies within the receptive field (in terms of Euclidean distance) of a unit other than the action that generated it, then it is this other action that is considered the relevant one in this situation. It is the Q-value of *this* state-action pair which is updated (along with its neighbours) and it is *this* unit which is moved through the motor space if the reward is greater than its current Q-value. In this way, large exploration of the Motor map can effectively result in increasing the exploration of the Q-learning algorithm. For example, the Q-learning algorithm may select action A_t say, but A_t may then produce a perturbed action that another Motor unit, say $A1_t$ actually takes responsibility for. And now $A1_t$ corresponds to exploration as far as the Q-learning algorithm is concerned since A_t was its original recommendation.

The reason this measure was found to be useful was that otherwise, when motor exploration is large, the situation may arise where Motor units and Q-values are taking responsibility for actual actions which are far from their own current jurisdiction, and for which there are more appropriate action units closer at hand. Also, it is desirable that the winning unit of the Motor map is indeed the one closest in weight space to the input that is going to drive the update of the map, otherwise topology preservation may be compromised. As the radius of exploration around the Motor units is annealed, the issue becomes less and less significant, since A_t will usually be the same as $A1_t$.

A second detail worth mentioning is that it may be an advantage to make exploration more consistent in the following way. If a state chooses to explore and selects an action unit with a Q-value that is not maximum for that state, then for the next few time-steps, if that state becomes active again (as it will tend to, because of the contiguous nature of the sensory input), then the same action as before is taken, irrespective of whether exploration or exploitation is recommended according probability p. Similarly if a Motor unit explores, then for some short time afterwards, if this Motor unit is given control again, the same perturbed action is taken. This will have the advantage of giving exploration a chance to make itself felt on the reward signal. This may be important since the time interval between actions may be small, and the same unit could otherwise take a number of different actions before a reward is noticed, inducing credit assignment problems. A 'window of consistency' equal to the horizon parameter h offers convenience and simplicity, and this is the approach used in these experiments.

A third detail refers to the way in which Output units are updated if any element of the target position in weight space lies outside the range $[0,1]$. In the experiments, the update was made according to the usual rule, and then the weights of the Output unit truncated to lie within this range. There are clearly other possibilities including passing the position of Output units in an infinite weight space through a squashing function (the sigmoid for example) to yield an actual action in finite action space.

Appendix C

Variances and Standard Deviations

$$\text{Variance of data } = \frac{\sum_{i=1}^{n}(x_i-\mu)^2}{n}$$

$$\text{Standard Deviation of data } = \sqrt{\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{n}}$$

$$\text{Standard Deviation of Mean } = \sqrt{\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{n(n-1)}}$$

where μ is the mean of the data, n is the number of data points, and x_i is the i^{th} data point.

Appendix D

Heuristics for setting parameters

The following section considers some simple heuristics for finding and coordinating a set of suitable parameters for the proposed model. While there is no precise formula for generating a suitable set, there are some empirical guidelines that may aid the process. The problem is made more complicated by the fact that the parameters are not independent, and interactions between these settings can make interpreting the results of different parameter sets difficult. Notwithstanding this, experience here has shown that providing a number of guidelines are followed, the exact values of these parameters is not important, and it is usually reasonably straight forward to arrive at a useful, working set of numbers. For convenience, the parameters are split into four categories: The Input map, the Output map, Q-learning and the reward function.

D.1 The Input map

- **Network dimensionality.** A network dimension of one or two was found to be sufficient for the simple problems considered so far. Too high a dimension may lead to a prohibitively large number of units being required. Lower dimension

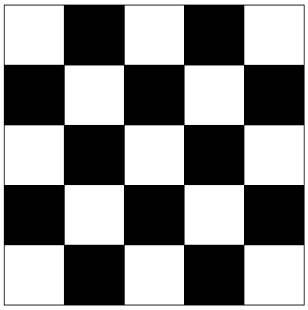


Figure D.1: A hypothetical input space with two classes of situations, each behaving differently under the reward function.

maps are easier to visualise and manipulate, and the hope is that the intrinsic dimensionality of the data is small. If the intrinsic dimensionality of the data is high, then strategies other than increasing the dimensionality of the map may be preferred, such as using a behaviour based decomposition of the problem. Then the problem can be decomposed to into a number of smaller mappings of fewer variables, and hence with lower dimensional state spaces.

- **Network size.** The number of units should reflect the variation in the reward function under all possible actions. Smaller networks are obviously favoured for efficiency, but large networks can always exploit topological preservation and neighbourhood learning if they turn out to be too big. Networks with between 20 and 100 units were found to be suitable for all the experiments performed here.

  Consider figure D.1 which shows a hypothetical input space with two classes of situations, with each class behaving differently under the reward function. Clearly at least 25 Input units are needed to adequately represent this problem, but in practice many more may be required to achieve an effective representational resolution. Care obviously needs to be taken when choosing a network size for both the Input and Output maps, and the designer should be as mindful as possible of the complexity of the mapping that underlies the task.

- **Neighbourhood size.** An initial neighbourhood the size of the entire network was often used to good effect. A minimum size of 1 or 2 was often imposed on the final neighbourhood size in order to encourage smoother maps. A zero neighbourhood (single unit updates) allowed units to 'wander' without respect for their neighbours with the consequence that small local kinks tended to develop in the map.

- **Neighbourhood annealing schedule.** The primary consideration when annealing the neighbourhood of the Input map is that it should be over a time scale that allows the agent to sense and respond to a representative sample of the input distribution. A further consideration is that because the input distribution may change with the Q-learning and the learning of new actions, care needs to be taken not to anneal the input neighbourhood before approximate actions and reasonable Q-value estimates have been formed. However, since it seems likely that the input distribution is generally less sensitive to changes in the rest of the learning system than the rest of the system is to changes in the Input map, there seems to be some argument for annealing the input neighbourhood quicker than the other learning parameters. The aim is then a semi-stable set of categories on which to base the remainder of the adaptation process.

- **Learning rate.** An initial learning rate of 1 was often used with success, annealed to 0 throughout the experiment, although smaller initial learning rates were also effective. The considerations for setting the learning rate are similar to those for the neighbourhood function. However, note that the plasticity[1] in the network is effectively reduced linearly with the learning rate and, in the case of a two-dimensional map, with the inverse square of the neighbourhood size[2]. Hence, if the annealing of the learning rate is linked to that of the neighbourhood size, an inverse cubic impact on overall plasticity is expected. Care must be taken not to leave the network short of plasticity. Common symptoms are 'stranded' units that are rarely active and very unfaithful representations of the input distribution. An annealing rate that satisfies the conditions of equation (2.14) is also preferred.

[1] We can define 'plasticity' as the total maximum potential for units to move through the space in a single time-step. Hence plasticity depends on learning rate and neighbourhood size.

[2] For example if the neighbourhood size parameter is halved, then the actual neighbourhood size in a 2D map is quartered etc.

D.2 Q-learning

- **Learning rate.** The learning rate, α, of the Q-learning algorithm was usually started at 1, and slowly reduced to 0. However, there may be some benefit in always maintaining a small amount of plasticity here so that the Q-learning process can continually fine-tune its mapping in response to small changes in the environment. The more noisy the reward signal, the lower α must be to generate a reliable estimate of expected reward. Note that noise will be generated by (amongst other things) large exploration in the Output map.

  It is important that there is still sufficient plasticity in the Q-learning process when the Input map is reasonably stable. This is because the expected reward of a state-action pair may change dramatically as Input units move through input space.

  For convenience, small, fixed Q-learning rates were often used from the outset in the experiments of chapter 6. However, the benefit of starting with high α is that rough estimates may be quickly found and then successively fine-tuned. This approach appears to expedite learning and was used extensively in most experiments of this thesis.

- **Exploration rate.** The purpose of exploration is to fuel the learning process, but it also generates noise. Optimising the exploration and learning rates of Q-learning in terms of this tradeoff is potentially difficult. A large amount of exploration leads to a noisy signal which in turn suggests a small learning rate. If the exploration is reduced, a higher learning rate may be effective, but there is now less fuel for the learning process. The familiar technique in Q-learning is to be safe and use small exploration and learning rates from the beginning — between 0.01 and 0.1 for example (Sutton and Barto, 1998; Araujo and Grupen, 1996). But reinforcement learning is a slow enough method anyway, and it is usually possible to speed up learning with judicious use of higher initial values for these parameters providing they are annealed appropriately. Unfortunately, there is no definition for 'appropriate' (beyond the theoretical criteria of equation (2.14)), and trial and error search must be employed for each individual case.

D.3 The Output map

- **Network dimension, network size and neighbourhood size.** The discussion is the same as for the Input map.

- **Learning rate, exploration, and the neighbourhood annealing schedule.** Parameterising the plasticity in the Output map is another difficult problem. The time it takes to acquire appropriate actions is dependent on the complexity of the reward function, the amount of exploratory noise in the system, the reliability of the signals from the environment, the learning rate of the Q-values, the complexity of the task, interactions between network size and neighbourhood size and so on. Unfortunately it seems that predicting how long it will take to learn a set of useful actions is practically impossible.

  However, it is likely that the formation of the Output map represents the bottleneck of the learning process, and this is useful to know. Learning the Input map can be very fast because it just has to react to the data it receives and does not have to go out and search for data itself. Although, as we have noted, this depends on being able to ignore the interactive effect of the other parts of the system on the input distribution, which may or may not be justified depending on the environment. Notwithstanding that all three modules of the system operate in parallel and interact with each other, it may be possible, at least in some circumstances, to crudely paraphrase the learning process sequentially in the following way:

  - Generate the Input map and in so doing define a set of working categories.
  - Repeat
    - Generate a new set of actions.
    - Evaluate these actions using Q-learning in the context of the categories formed earlier.
    - Keep the actions that do better than their predecessors.

  The suggestion here is that the generate-and-test approach to discovering new actions will be the slowest part of the system, particularly as the iterated 'test' part of this process involves Q-learning which is itself a slow process. This indicates that the time it takes to form the Output map should be used as a guide for the parameters of the rest of the system. Again, trial and error is required. Specifically, the neighbourhood, learning rate and exploration rate of the Output map should be started high and annealed quicker and quicker over a succession of runs until a reduction in final performance is observed.

A further implication of the iterated-test role of Q-learning is the suggestion that the Output map should only be allowed to change slowly with respect to the Q-values. If this is not respected then the system may become swamped with noise as Q-values lag behind newly discovered actions, become inaccurate, and then allow Output units to move spuriously through the output space.

One further point worth mentioning is that the analysis of the formation of the Output map showed that the exploration parameter, MA, can have a dramatic effect on the type of regions within the output space that are favoured. For example, Figure 6.14 showed how a transition from a global reward assessment to a local hill-climb affected the position within the action space that A_1 was attracted to. The conclusion is that if the reward surface is known to have no local minima, a small exploration value can be used more efficiently from the beginning. Otherwise, a large initial value of MA is recommended.

D.4 The reward

The key parameters pertaining to how the reward information is used are the horizon, h, and the discount factor γ. The horizon should be chosen to reflect the maximum delay of a reward following an action. The more immediate the reward, the lower h can be and the faster learning can proceed. If classical Q-learning is used, an implicit infinite horizon is adopted, leaving the discount factor the only parameter to set. Again, the issue is how delayed the rewards are likely to be. The smaller γ, the more weight is given to immediate reward information. If Q-learning with eligibility traces is used, then the additional parameter λ must be considered. Following the discussion of Sutton (1996), $0 < \lambda < 1$ is likely to be optimal, and this is borne out by Tesauro (1992) who used intermediate values of λ to good effect in his *TD-Gammon* application.

D.5 Dynamic environments

This discussion of parameters assumes that the environment is stationary and that the reward function and input distribution change either not at all, or at least very slowly. If this is not the case, then we have to consider the continued plasticity of the system. Unfortunately there is more to continual adaptability than just maintaining small learning rates, neighbourhoods, and exploration. In particular we have seen how the *MA* parameter of the Output map, which controls the exploration around each Output unit, affects the system's vulnerability to local minima. If the environment is dynamic, then it is possible that parameters such as this will sometimes need to be reset to their initial and most plastic states to avoid new local minima at a cost of a large amount of noise to the rest of the system.

Highly dynamic environments suggest dynamic parameters that are linked to the environment and not fixed to the age of the agent. But this presents a significant challenge since not only is it impossible to know when actions are sub-optimal and therefore when to increase exploration, but there is also no way of knowing how much plasticity should be restored, and for how long. If the environment is dynamic, then because the method is a sampling one, the system can never rule out the possibility that some currently uninhabited part of the Output space is worth exploring.

There are a couple of changes to the basic architecture that could allow the system to adapt to a highly dynamic environment. The first is to invoke short periodic bouts of exploration, long enough to implement a brief reconnaissance for better actions but short enough to avoid seriously disrupting the system. The second is to use the error in the Q-values as an indicator that the environment is changing. However linking exploration to this error must be moderated so as to avoid runaway feedback loops. A third approach is that of the ART network (Carpenter and Grossberg, 1987a), which addresses the stability/plasticity dilemma by ensuring that radical plasticity takes place in a fresh part of the system, and does not disrupt existing behaviour. This may have biological precedents. For example, Wilson and McNaughton (1993) suggest that new spatial information has little effect on previously stored stimuli with respect to a rat's navigational capabilities.

Within the context of the Output map this discussion suggests the use of a constructive SOM which has the ability to add units dynamically. However, the implications for topology preservation are not clear. If the system can *always* learn new information, then assuming its resources are finite, it must also be able to forget obsolete information. This is not necessarily a trivial problem either. It is interesting to note that as infants we seem generally able to acquire new information very rapidly, relatively permanently, and with minimal disruption to existing knowledge. This is somewhat analogous to the ART network. However, in later life, learning not only appears to be slow, but new skills seem to interfere with existing ones and are easily forgotten if not reinforced. This is behaviour more familiar to the SOM.

In summary, highly dynamic environments can be dealt with in the Input map and Q-learning modules by maintaining non-zero learning and exploration rates, with a small performance cost incurred. Dealing with these types of environments in the Output map is not so straight forward and requires significant further research.

D.6 Parameter conclusions

To simplify the problem of setting such a large number of parameters, a single annealing rate, $f(t)$, was used for all parameters in the main experiments of this thesis. All that was varied were the starting constants, from which all parameters were then decayed at the same rate. No analysis was performed to determine whether this represents an optimal strategy (it seems unlikely), but satisfactory results were obtained nonetheless. If finding suitable annealing schedules becomes problematic, then using a set of small, fixed learning rates may provide a solution. There may be a cost in performance, and of course annealing schedules for the neighbourhoods and exploration still need to be discovered.

As a last and speculative thought on annealing rates, it is interesting to imagine that there might be a good reason for why our own ability to learn appears to diminish with age. Although popularly assumed that becoming less adaptable with age is a symptom of pathological degradation, there may be a more positive explanation. We have seen how learning is a noisy and disruptive process. Fuelled by exploratory behaviour, adaptation incurs a cost that is less justifiable the further through a finite lifetime an agent progresses. In the context of the experiments performed here, in order to hone existing skills, both the learning rate and exploration had to be reduced further and further, and in step with each other. The loss of the human ability to learn with age could be interpreted as an explicit attempt by evolution to maximise the area under the performance curve of its agents.

Appendix E

Backprop experiment (Netlab script)

The following code was run in Matlab (version 5.3.1.29215a (R11.1)) to test a back-propagation trained network's ability to learn an oscillating mapping in which small changes in the input repeatedly generate the largest possible changes to the output.

The experiment was performed for various learning rates and hidden layer sizes, but the mapping was never achieved satisfactorily.

```
%clear;
close;

NO_INPUTS = 1;
NO_OUTPUTS = 1;
NO_HIDDEN = 56;
LEARNING_RATE = 0.01;
MAX_EPOCHS = 10000;
AVERAGE = 1000;
```

```
START = 0;
END = 10;
R = 0.01;

I = 1:NO_INPUTS;
O = 1:NO_OUTPUTS;

FUNC = 'linear';

NET = mlp(NO_INPUTS,NO_HIDDEN,NO_OUTPUTS,FUNC);

sum = 0;
count = 0;

% Training
for INDEX=1:MAX_EPOCHS

I(1) = (rand*(END-START))+START;



if I(1)>=0 & I(1)<=1 T(1) = 0.5; end
if I(1)>=1 & I(1)<=2 T(1) = 0.0; end
if I(1)>=2 & I(1)<=3 T(1) = 0.5; end
if I(1)>=3 & I(1)<=4 T(1) = 0.0; end
if I(1)>=4 & I(1)<=5 T(1) = 0.5; end
if I(1)>=5 & I(1)<=6 T(1) = 0.0; end
if I(1)>=6 & I(1)<=7 T(1) = 0.5; end
if I(1)>=7 & I(1)<=8 T(1) = 0.0; end
if I(1)>=8 & I(1)<=9 T(1) = 0.5; end
if I(1)>=9 & I(1)<=10 T(1) = 0.0; end

O = mlpfwd(NET,I);
```

```
e = abs(O(NO_OUTPUTS)-T(NO_OUTPUTS));
sum = sum +e;


if (mod(INDEX,AVERAGE) == 0)

  INDEX

  count = count + 1;
  Error(count,1) = INDEX;
  Error(count,2) = sum / AVERAGE;
  sum = 0;



  % Draw Response
  figure(2);
  hold off;
  plot(0,0);
  hold on;

  % Plot network response
  for II = START:0.1:END

    I(1) = II;
    OO = mlpfwd(NET,I);

    % Generate Circle

            TT = 0:pi/50:pi*2;

            X1 = sin(TT)*R + II;
            Y1 = cos(TT)*R + OO;
```

```
    C1 = [1 1 1];

            h = fill(X1,Y1,C1);
    %h = fill(I1,O1,C1);
            set(h,'Clipping','off');

          end

  axis([START END 0 0.5]);
  drawnow;

end

G = mlpgrad(NET,I,T);

WEIGHTS = mlppak(NET);
WEIGHTS = WEIGHTS - (G*(LEARNING_RATE));

NET = mlpunpak(NET,WEIGHTS);

end

figure(1);
plot(Error(:,1),Error(:,2))
axis([0 MAX_EPOCHS 0 1])

axis xy
axis equal
set(gca,'FontSize',20);
axis([START END -1 1]);
set(gca,'XTick',[0:0.5:1]);
set(gca,'YTick',[0:0.5:1]);
set(gca,'Color',[0.8 0.8 0.8]);
```

```
set(gcf,'Color',[1 1 1]);
xlabel('Input');
ylabel('Output');

box on;
```

Appendix F

Abstract Categories

This chapter represents some preliminary investigation into abstracting over the representations generated by the proposed model of chapter 5. The ideas in this chapter are largely stimulated by Karmiloff-Smith's psychological account of child development in which she proposes a step-wise developmental process based on 'representational redescription' (Karmiloff-Smith, 1995). Her thesis emphasises the importance of using the right symbols to represent the world, and hypothesises an iterative, incremental and interactive process for achieving this. The work of this appendix is somewhat tangential and also lacks an application; it therefore does not form part of the thesis proper. However, it is included as an appendage because it is considered interesting and potentially useful.

F.1 The problem

Consider again the experiment presented in chapter 5 in which a Khepera robot learns to avoid obstacles. Figure F.1 is a reminder of typical learned formations of the Input and Motor maps. Recall that the input space was six dimensional and the motor space two dimensional. Figure F.2 is a reminder of the sensor and motor labelling. The first

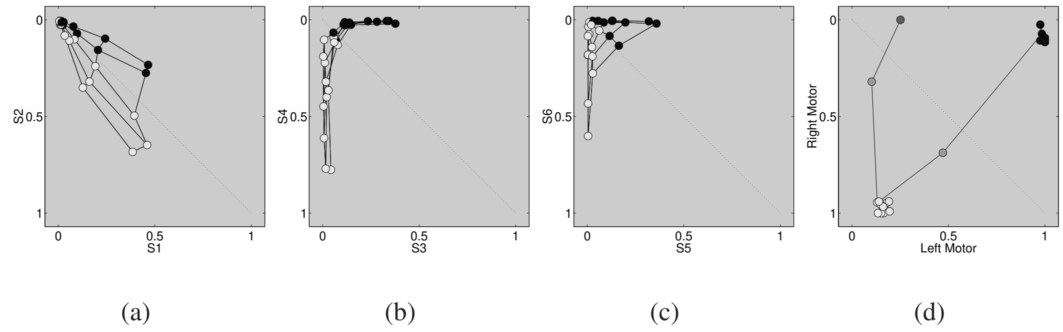


Figure F.1: The Input and Motor maps after learning. Duplicated from figure 5.19.

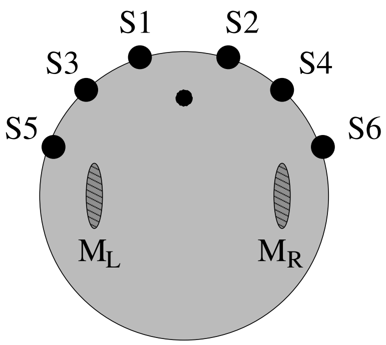


Figure F.2: Sensor and motor labelling on the robot.

observation is that although there are twenty units in the Motor map, most of these units have clustered around just two positions, one corresponding to a hard left turn on the spot, and one to a hard right turn on the spot. It seems these were the only two actions required for efficient obstacle avoidance. Each unit of the Input map is coloured according to the Motor unit to which it has the strongest connection (or highest Q-value estimate). All the units above the $\langle 0,0\rangle$-$\langle 1,1\rangle$ diagonal line correspond to situations with more left sensor activity than right, and units recognising these situations propose right turns. The region below the diagonal line represents situations with more right sensor activity than left, and units recognising these situations propose a turn in the opposite direction. Hence the agent has learned to turn away from the side with the most sensor activity with one of two actions.

Implicitly, the system has cleaved the six-dimensional state space into two halves, with units in each half proposing either the same action, or at least actions that are very similar. There now seems to be some potential benefit in looking for a way to make this acquired knowledge *explicit*. An abstract representation of the input space could then be boiled down to just two classes rather than the twenty-five classes currently comprising the Input map. One benefit of a more compact representation is that if future tasks could re-use these abstract categories, learning may proceed much more quickly. For example, if the representations could be condensed to just two input space categories and two output space categories, there would be just four Q-values to discover instead of the 500 used in the original experiment. Of course there is no point in re-learning the same task, but future tasks that are related to the original problem could clearly benefit.

It makes sense to start with a simple problem, so consider the following task: The input space is the unit square, with inputs occurring randomly and uniformly throughout the space. The output space is also the unit square. The system receives a reward negatively proportional to the Euclidean distance between its *actual* output and the *desired* output, where the desired output for the region of the input space above the $\langle 0,0\rangle$-$\langle 1,1\rangle$ diagonal is the $\langle 1,0\rangle$ corner of the output space, and the target output of the input space below the diagonal is $\langle 0,1\rangle$. Reward is immediate and so an horizon of 1 is used. In this way the setup is similar to the latter experiments of chapter 5 where a desired, target output for each input is known, and a scalar reward given according to how close the system's actual output is from this optimum action. As we discussed in chapter 5, this still constitutes a simple form of reinforcement learning because the targets are not provided explicitly; knowing the distance to the desired output does not reveal the direction in which the output lies.

Figure F.3 illustrates how the mapping to be learned is just a simplified version of the obstacle avoidance behaviour.

To simplify matters further we can remove the need to learn the outputs by hardwiring the two target actions. Now all the system has to do is learn the value of taking each of the two fixed actions at each point in the unit square of the sensory space. So if a left turn is prescribed, the system will receive a reward of zero for selecting the left turn action, and $-\sqrt{2}$ for selecting the right turn action, and vice-versa.

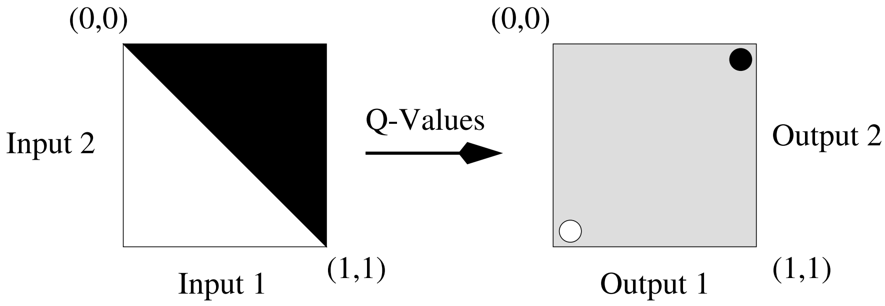
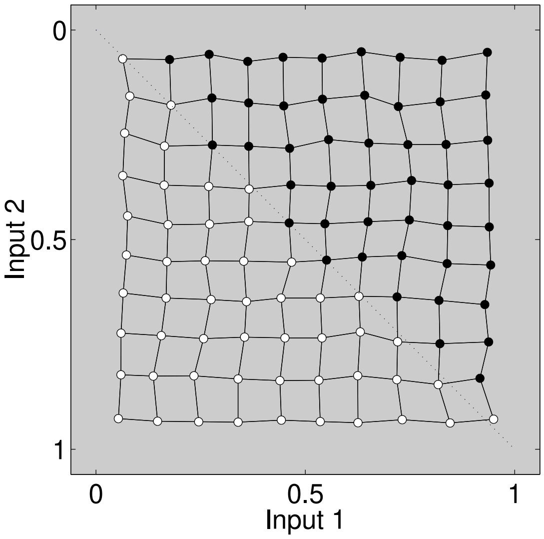


Figure F.3: The simplified obstacle avoidance problem. The input space is just two-dimensional and actions are rewarded according to their distance from one of two target actions.

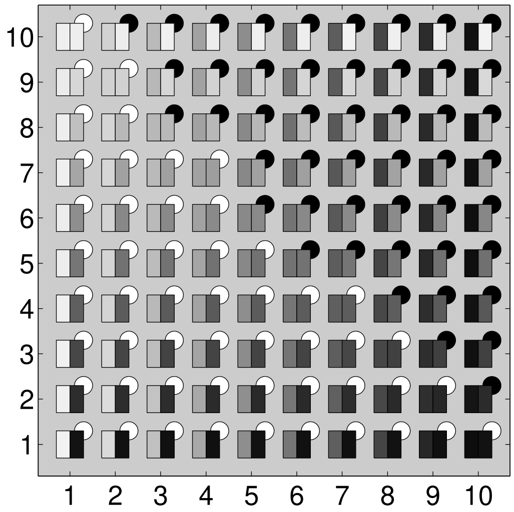
Figure F.4 shows the Input map after learning. The familiar plot in (a) shows the map in the input space with each unit colour coded according to which of the two Output units it has the highest Q-value for. Figure (b) shows the same map but in the topological or physical space of the network. Each unit has two bars associated with it — one for each input dimension — showing the prototypical situation responded to by that unit (black = 1, white = 0). The circle in the corner of each unit again shows the Output unit to which that Input unit is most strongly connected.

So the desired mapping is learned for this simple task. Now the aim is to dynamically and autonomously construct a number of abstract categories for the input space which will reflect the natural dichotomy of this space, thus allowing a more compact representation. The general problem can be condensed to that of grouping Input units into classes, with the members of each class yielding similar Q-values under similar actions. In the example above, there appear to be two distinct classes, but in general the number of classes will not be known.

First we consider some desirable model properties. How the Q-values are calculated should be of no relevance to these abstract categories. For example, the parameters such as the horizon and learning rate, and even the method of value estimation itself (e.g. Q-learning, SARSA, Monte-Carlo averaging) should be *encapsulated*. The formation of the abstract categories should then depend only on the current ‘Q-values’. However, given the nature of these estimates and the process that generates them, the formation of abstract categories should be an iterative and incremental process that is robust to these values being noisy and that allows for these values changing over time.



(a) Input map in weight space after learning



(b) Input map in physical space after learning

Figure F.4: The Input map after learning the simple problem of figure F.3.

There is one other point. It is not sufficient to simply group Input units that prescribe the same action, because in general there could be any number of Action units occupying similar regions of the Output space (as in figure F.1(d) for example). This would preclude efficient generalisation.

F.2 A solution

We now have a working specification of the problem. The following definition is now given:

Two Input units (S and T) are similar if and only if they yield similar Q-values for every action. i.e. $Q(S,a) \approx Q(T,a)$***, for all a.***

Consider an Input map containing three units and an Output map containing four units.

All the Q-values of the system can then be represented as a 3x4 *Q-matrix*[1]:

$$\begin{bmatrix} a1 & a2 & a3 & a4 \\ b1 & b2 & b3 & b4 \\ c1 & c2 & c3 & c4 \end{bmatrix}$$

with element (m,n) representing the Q-value between the mth Input unit and the nth Output unit. Now the *Q-vector*:

$$\begin{bmatrix} a1 & a2 & a3 & a4 \end{bmatrix}$$

defines the Q-values from the first Input unit to each of the four Output units. According to the assumption above, the second Input unit will be a candidate for belonging to the same class as the first Input unit if and only if:

$$\begin{bmatrix} b1 & b2 & b3 & b4 \end{bmatrix} \approx \begin{bmatrix} a1 & a2 & a3 & a4 \end{bmatrix}$$

More generally, if we plot each row of the Q-matrix in n dimensional space, clusters in this space should correspond to classes of 'similar' Input units, under the definition given. This turns the problem of identifying abstract categories into one of clustering in n dimensional space. For consistency with the approach taken so far, another Kohonen map could be used to map this space and discover these clusters, with each unit in this higher order map now representing a prototype for an abstract category. Since the aim is to keep the number of abstract categories low, only a small one dimensional SOM is proposed. As it happens, the topology preserving nature of the SOM is of no immediate use here, and other clustering algorithms could be substituted in its place. However, the incremental update rule is a useful feature because the abstract classes must continually respond to changes in the underlying Q-values. The topology preserving nature of the map may transpire to be useful later on, in the same way that topology preservation in the Input and Output maps turned out to be an asset in previous experiments.

[1] Also called the Q-table.

Figure F.5 illustrates the augmented system. In this example, there are two inputs to the system which connect to a 3x3 Input map. There are also two outputs from the system which are connected to a 4x1 Output map. The connections between the two maps represent the Q-matrix. All this is familiar. However, a new *Abstract Input map* comprising just two units is now added to the system. This extra map has four inputs, one for each unit in the Output map. Each unit of the Input map now not only connects to each unit of the Output map, but also to the inputs of the *Abstract Input map*. As the Abstract Input map self-organises on the vectors of Q-values associated with each unit of the Input map, its two units should be attracted to the centre of clusters and thus learn to represent prototypes for classes of Input units with similar behaviour.

One way to view the results of running this system on the simplified problem discussed above, is to label the entire input space according to which abstract category it belongs. This is a two-step process. First, for each point in the unit square, the winning Input unit is identified. Recall that this is just the unit with the smallest Euclidean distance between the input vector and that unit's weight vector. Having identified the winning Input unit, the vector of Q-values from that unit to each Output unit is then retrieved. In the case of the matrix of Q-values given above, this will be either $\left[\begin{array}{cccc} a1 & a2 & a3 & a4 \end{array}\right]$, $\left[\begin{array}{cccc} b1 & b2 & b3 & b4 \end{array}\right]$ or $\left[\begin{array}{cccc} c1 & c2 & c3 & c4 \end{array}\right]$. Secondly, this vector is applied to the Abstract Input network, and the winning unit of *this* network (again with the smallest Euclidean distance) is declared the abstract category to which the original input stimulus belongs. The two stages of the process reflect the fact that Input units categorise input stimuli, while Abstract units categorise Input units.

Figure F.6 shows the input space labelled in just this way. Since there are only two units in the Abstract Input network, and therefore just two classes, each point is shaded one of two colours. For convenience, black and white are the two colours chosen. The satisfying result is that the input space has been dichotomised into two regions, each containing stimuli that behave consistently with respect to the estimated expected return under each action.

A second way to view the effect of the Abstract map, is to look at the *degree of membership* of each input stimulus to each abstract category. For an input stimulus, I, and an abstract category, c, we can calculate the degree of membership of I to c using the fol-

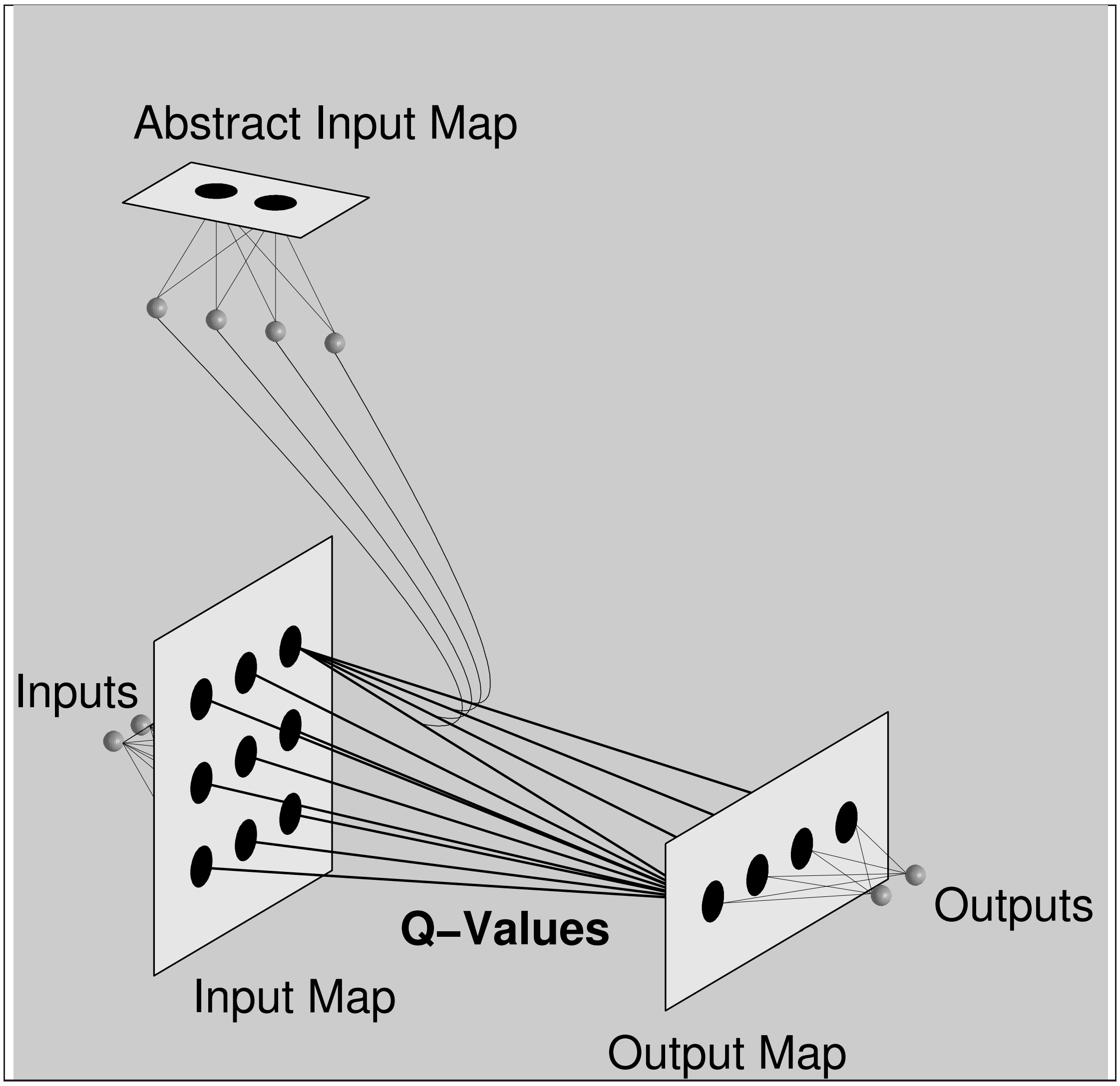


Figure F.5: An illustration of the augmented architecture. The Input and Output maps are implemented as before and the Q-values are maintained between the two maps in the usual way, although not all the connections are shown. A small *Abstract Input map* consisting of just two units has four inputs which correspond to the four units in the Output map. Each unit of the Input map not only connects to each unit of the Output map, but also to the inputs of the *Abstract Input map*. The input to the Abstract map is generated as a direct result of activity in the Input map. Hence learning in the Abstract map takes place parallel to learning in the rest of the system.

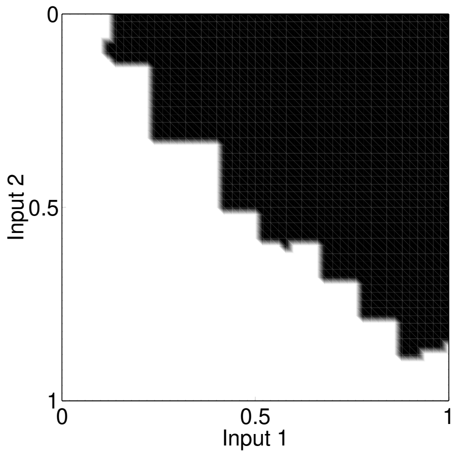


Figure F.6: The input space is coloured according to the abstract category to which each point belongs.

lowing procedure: First identify the winning Input unit for I — let us call this s — and then the corresponding Q-vector for s, i.e. $\begin{bmatrix} a1 & a2 & a3 & a4 \end{bmatrix}$, $\begin{bmatrix} b1 & b2 & b3 & b4 \end{bmatrix}$ or $\begin{bmatrix} c1 & c2 & c3 & c4 \end{bmatrix}$. Now calculate the Euclidean distance between this Q-vector and the weights of Abstract unit c. Finally, normalise this distance to the range $[0,1]$ so that the largest such distance for any I and c yields a membership of 0 and the smallest such distance yields a membership of 1. This last step allows the results to be visualised easily. Figure F.7 shows the degree of membership of each possible input stimulus to each of the two abstract classes that were learned. The plots show clearly how each Abstract unit learns to identify a distinct yet consistent half of the input space.

The Abstract Input map introduces some new parameters, including the network size, the initial neighbourhood size, learning rate, and neighbourhood annealing schedule. In the experiment above, the following values were used. The mnemonics refer to the same parameters as described in previous chapters:

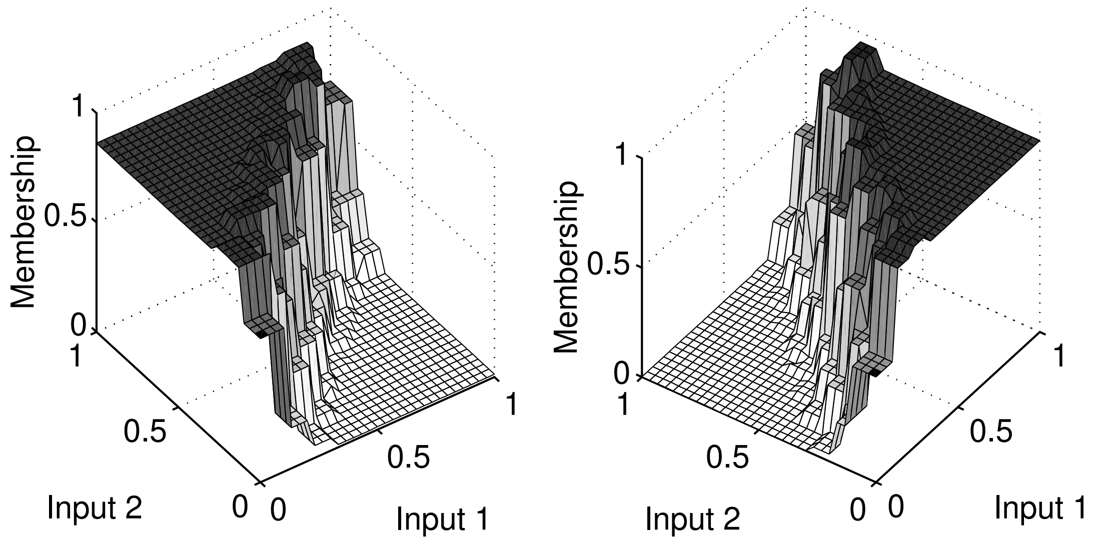
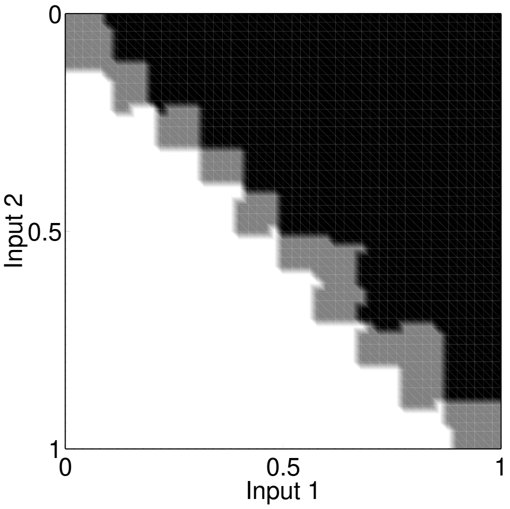


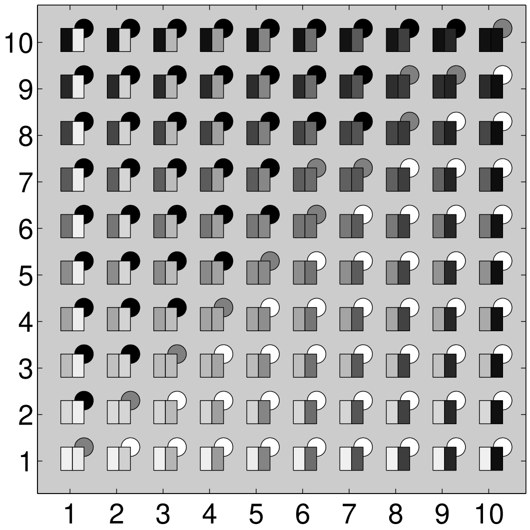
Figure F.7: Membership of each point in the input space to each of the two abstract categories. Each point is colour-coded according to the degree of membership.

Parameter	Value
Abstract network size	2×1 units
Neighbourhood size	$4 \times f(t)$
Learning rate	0.01 (fixed)
Minimum Neighbourhood	1
Minimum Exploration (MA, p)	0.1
Minimum Learning rates (IL, OL)	0.2

$f(t)$ is annealed in the usual way ($f(t) = 0.9998^t$) but note that MA and OL are redundant in this experiment because there is no learning in the Output map. No quantitative results are given because the time scales for learning are identical to those seen for similar tasks in chapter 5. The formation of abstract categories has not added to the learning complexity since abstract category formation takes place parallel to the formation of the Input map and the estimation of Q-values.



(a) Categorised input space. Regions belonging to the third unit are shaded grey. The third unit categorises the region where the other two regions meet



(b) Physical map of the network. The third class emerges along the diagonal. Note that the Input map is oriented differently in the input space to before. The orientation depends on the initial random configuration.

Figure F.8: Categorisation of the input space and the Input map after learning. The third Abstract unit is employed along the joining diagonal.

F.2.1 Increasing the number of abstract categories

In the previous section the network was given exactly the number of abstract categories that were assumed to be relevant, but it is interesting to see what happens when the number is increased. Figure F.8 shows how the Input map is categorised after learning the same problem but using three Abstract units instead of two. Interestingly, the third Abstract unit is employed along the diagonal that separates the previous two regions. Figure F.9 confirms this from the perspective of class membership.

The reason for the third category representing the diagonal is straight forward. All Input units responding to the region of input space below the diagonal will yield a reward of 0 for taking the first action, and $-\sqrt{2}$ for taking the second action. The converse is true for all the units of the Input map responding to the region above the diagonal.

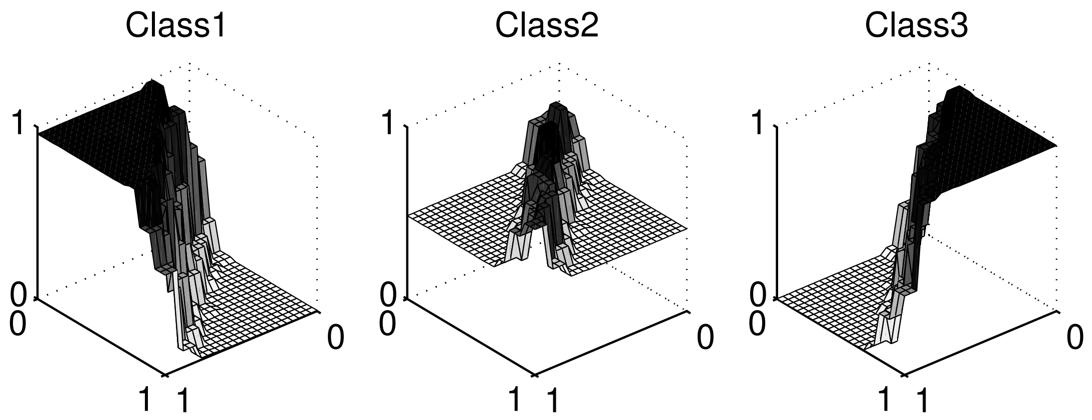


Figure F.9: Membership of each point in the input space to each of the three abstract categories.

Hence we expect each Input unit to have a Q-vector of $[0 \quad -\sqrt{2}]$ or $[-\sqrt{2} \quad 0]$. However, units along the diagonal will have to respond to situations both just above *and* just below the diagonal, yet with a consistent action. This introduces perceptual aliasing. A unit actually *on* the diagonal will receive a reward of 0 half the time and a reward of $-\sqrt{2}$ the other half, irrespective of which action that unit prefers. So the Q-vectors of Input units along the diagonal will tend towards $[-\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}]$, thus creating a third cluster of Q-vectors in two-space. When the number of abstract categories is increased to three, this relatively small third cluster can then be represented. Note that all regions of the input space seem to have *some* membership to the 'diagonal' category. This is because the Q-vectors $[0 \quad -\sqrt{2}]$ and $[-\sqrt{2} \quad 0]$ are half the distance from the Q-vector $[-\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}]$ as they are from each other. This state of affairs is satisfactory because each Input unit behaves like any particular diagonal unit half of the time, and therefore has some claim to belonging to the diagonal abstract class.

So what happens if too many abstract categories are used? Figure F.10 again shows the input space coloured according to which abstract class it belongs, but this time after learning with six abstract classes. On first inspection it appears that four out of the six have learned the diagonal class, but the membership graphs of figure F.11 show that this is misleading. In fact only the fourth Abstract unit represents the diagonal class, and all the others represent either the region above the diagonal or the region below. This behaviour is both predictable and reasonable for the following reason. Since out

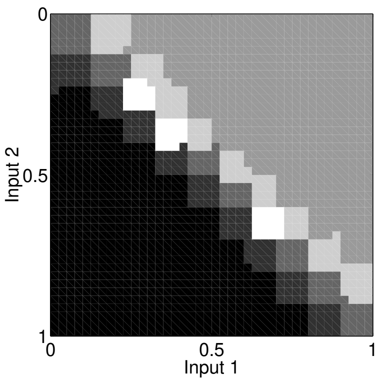


Figure F.10: The input space is coloured according to which of six abstract classes it belongs. White and black, and four shades of grey are used.

of one hundred Input units only about ten lie on the diagonal, the Abstract Input map preferentially represents the other two regions, and this is more evident in the situations where there is redundancy in the number of abstract classes available.

Notice how units three and five have a characteristic shape that is somewhere between the diagonal class and their other neighbour. This is because a minimum neighbourhood of one is maintained in the Abstract map, and these units get 'strung out' between their two neighbours. Annealing the neighbourhood all the way to zero would result in these units becoming completely stranded. There always has to be at least one such unit between any two well defined classes, as long as there are redundant units at that position in the network.

It is also worth noticing that topology preservation is present in the abstract classes. Classes that are most similar are neighbours. It is possible that this could be exploited in further developments to the architecture.

F.2.2 Learning the Output map

So far just two Output units were fixed in the output space at suitable positions, and the goal of the system was to learn to choose between them. Since one of the aims of this thesis is to enable learning within the action space, another experiment is now

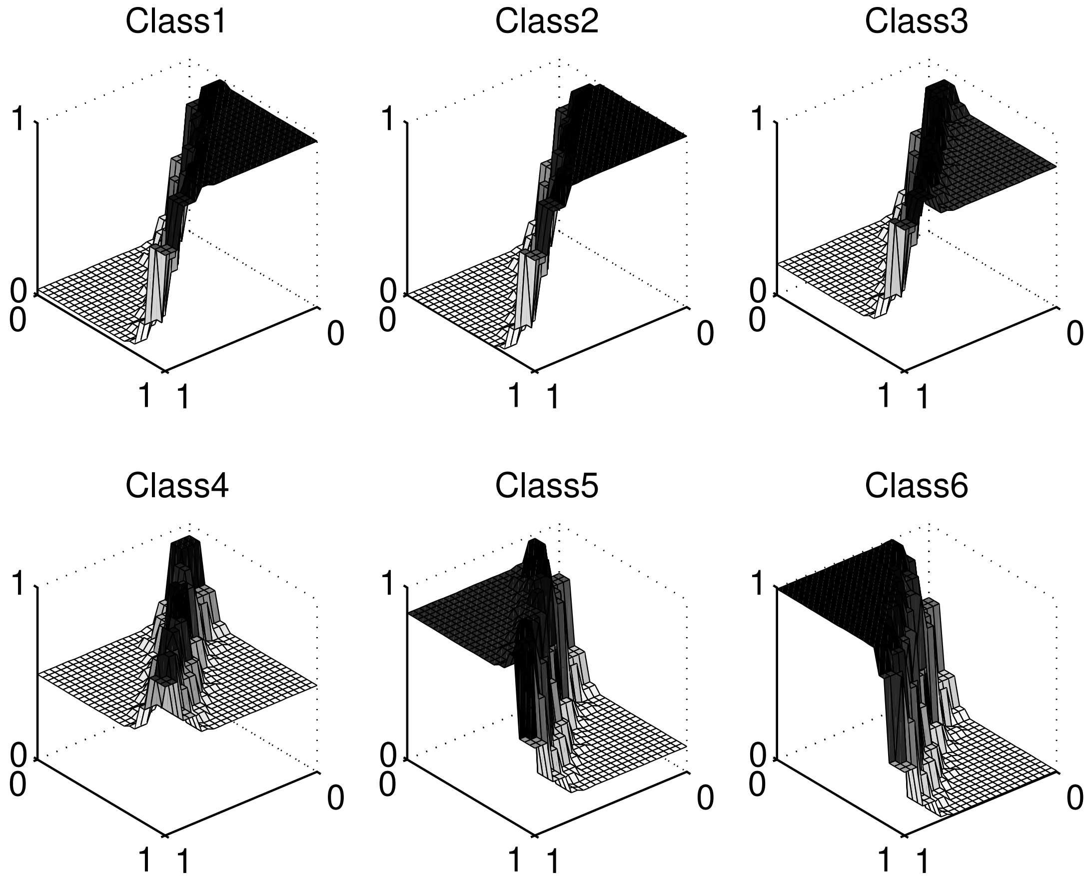
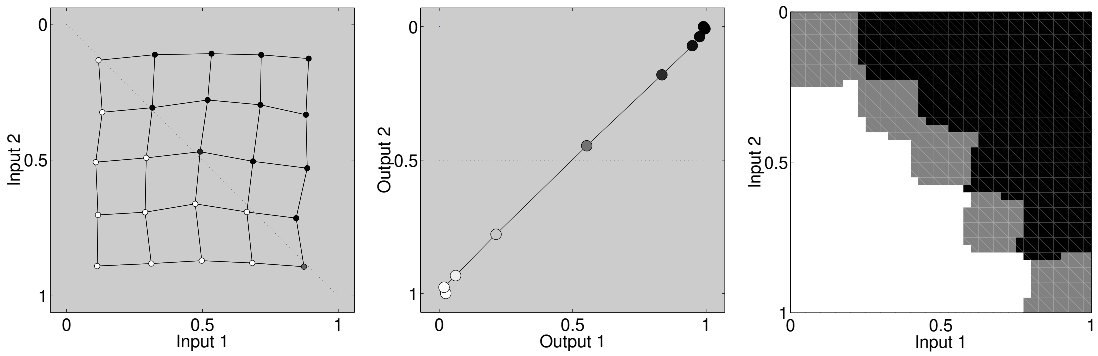


Figure F.11: Membership of each point in the input space to each of the now six available abstract categories. Note the topology preservation in membership profiles.

performed in which the two desired actions corresponding to the points $\langle 0,1\rangle$ and $\langle 1,0\rangle$ in the output space must be discovered rather than being hardwired. The same reward signal is used and the abstract mapping takes place exactly as before. However, the Abstract map now has to form over moving Q-values which are themselves estimates of the expected return over moving states *and* actions.

For this experiment, ten Output units connected in a one dimensional Kohonen map, and a five by five two dimensional Input map are used. The Abstract network contains three units. The results are predictably similar to the previous experiment, and the only difference is that there are now essentially four parallel learning processes: Formation of the Input map, learning a set of suitable actions, estimating the return of taking each



(a) Learned Input map, colour coded according to the Output unit with the highest Q-value.

(b) Learned Output map comprising ten units.

(c) Categorised input space.

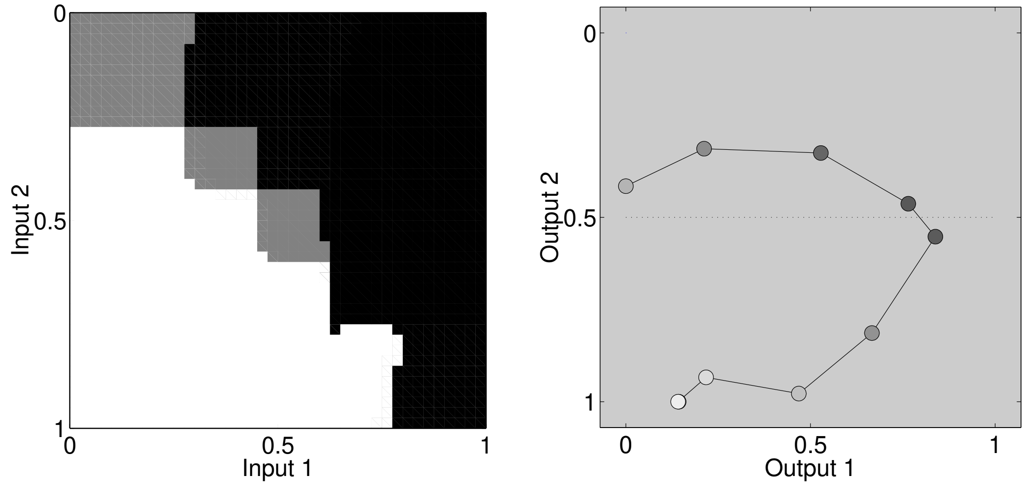
Figure F.12: The Input and Output maps, and categorisation of the input space after learning. As before, the third unit is employed along the joining diagonal.

action in each state, and abstracting over the Q-values to form the abstract classes. Figure F.12(a) and F.12(b) show the learned Input and Output maps, colour-coded in the usual way, and F.12(c) shows the input space coloured according to abstract category membership. Apart from demonstrating that the abstract classes still form despite the Abstract map now having a noisy ten dimensional[2] clustering problem instead of the previous neat two dimensional problem, figure (b) also illustrates the unit 'stranding' effect just mentioned. There are now three units out in no man's land because the minimum neighbourhood size used in the Output map for this problem happened to be two.

F.2.3 Partial learning

It may seem as if the four learning processes mentioned above each have the accurate results of the previous process as a prerequisite for correct performance. For example, it may seem that the Input map ought to be formed before the output space is explored, and that the Q-values need to be accurate before the abstract classes can be constructed.

[2]Because there are now ten Output units.



(a) Partially learned abstract categories.

(b) Partially learned Output map.

Figure F.13: The abstract categories can be formed in parallel with the optimisation of the rest of the system. At this early stage of learning, the regularities of the input space are close to being discovered despite the fact that the Output map is a long way from optimality and the Q-values still contain a large amount of noise generated by exploration.

But in the spirit of the 'bootstrapping' learning paradigm this is not the case, which is fortunate since there tend to be circular dependencies. In particular, the abstract categories can be forming even while there is noise, error, movement and exploration in the rest of the system. This means that the abstract categories can be approximated at a very early stage. An illustration of this is seen in figure F.13 in which plot (a) shows the abstract classes almost completely formed at a point so early on in training that the Output map (b) is still a long way from optimality. In fact the annealing parameter, $f(t)$, is approximately 0.5 at this point which means there is still a huge amount of exploratory noise in the system. Although the final behaviour of the system is still some way off, the Abstract Input network has still abstracted most of the regularity imposed on the input space by the task and the reward signal.

For completeness, figure F.14 shows the formation of the three categories as a sequence of snapshots taken during learning.

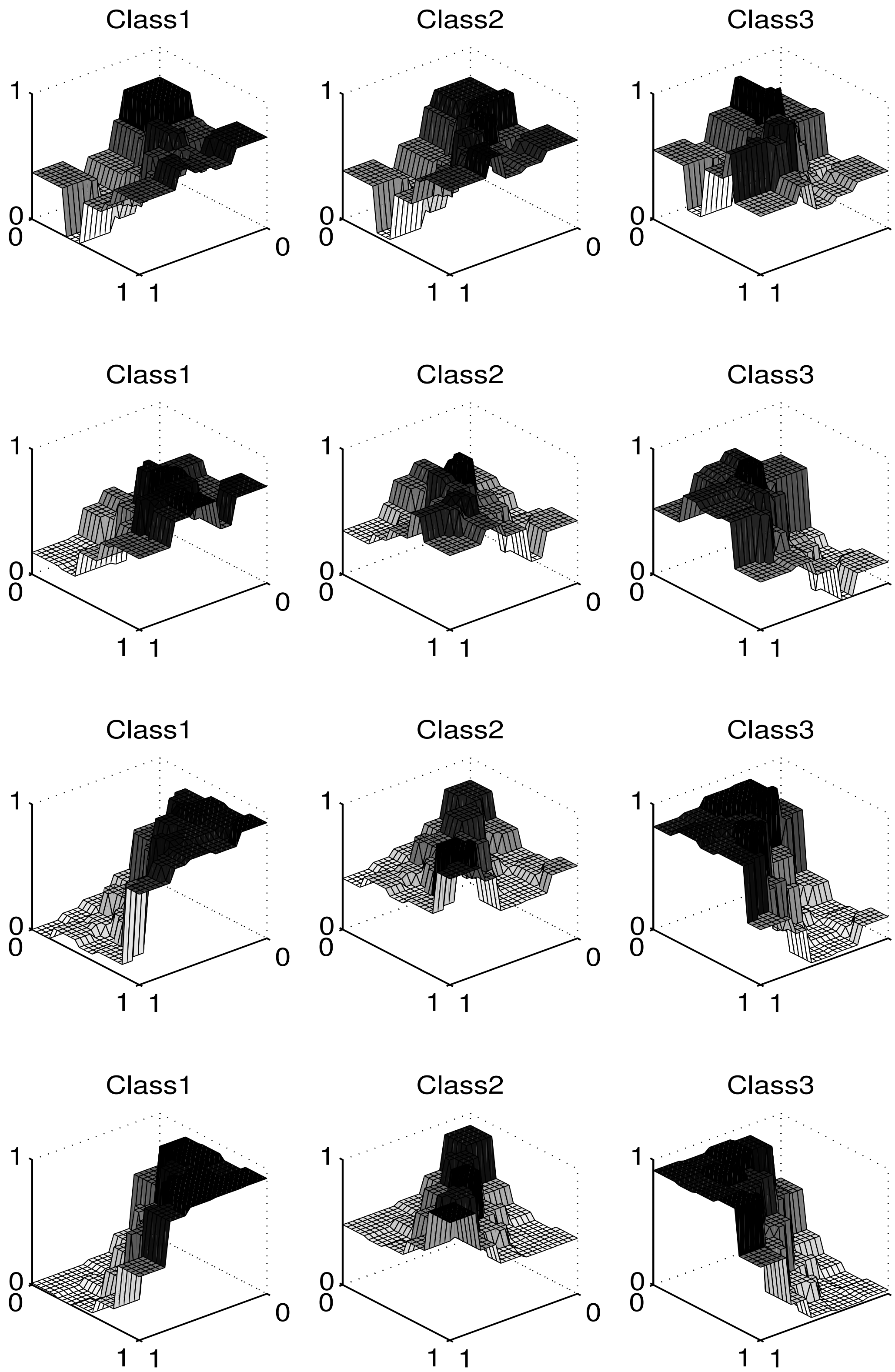


Figure F.14: Formation of the abstract categories at $t = 1000$, 2500, 10000 and 20000, corresponding to $f(t) = 0.8, 0.6, 0.15$ and 0.02.

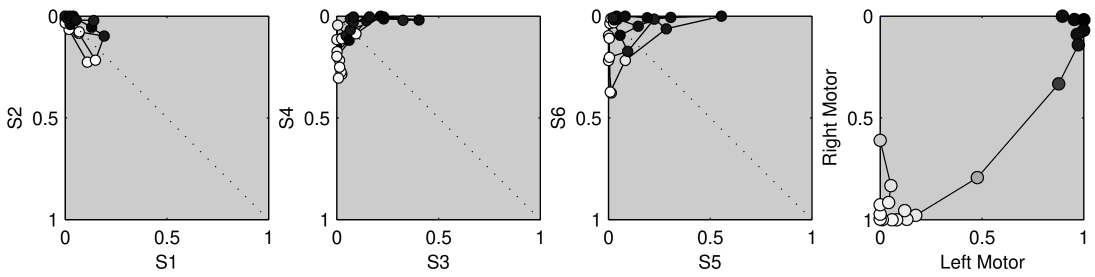
F.2.4 Avoiding obstacles

The problem of learning abstract categories is now tackled for the experiments involving the simulated Khepera robot learning to avoid obstacles. The experimental setup, including the parameter settings are the same as described in section 5.6. The usual six sensors are used as input to the system, and as before there are two motor outputs.

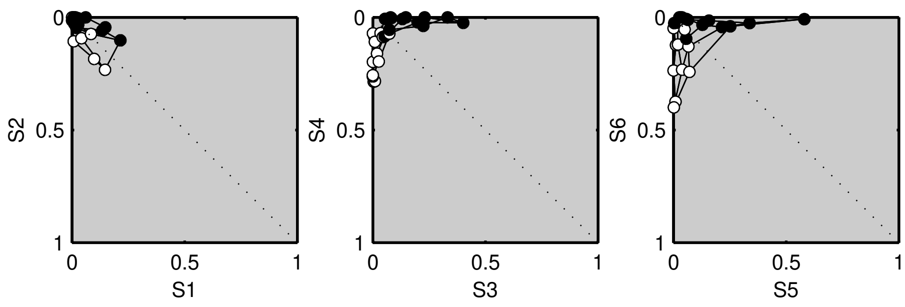
The problem of learning to avoid obstacles is harder than those encountered so far because the clusters of Q-vectors will not be well defined. For example, even all the Input units responding to an abundance of left sensor activity and proposing a right-turn action will generally have different Q-values for that same action because of the possible variation in the sensor readings. There is now a continuous range of situations, actions and discounted rewards, and there will no longer be identical Q-vectors with sharp, well-defined cluster boundaries.

The first experiment is performed with just two Abstract units in the hope that the input space can be dichotomised in the same way as before. The results, shown in figure F.15, suggest that the abstraction process is indeed able to do this. Figure F.17 shows the same information as figure F.15, but in the physical space of the Input network. The left plot shows the Input map with each Input unit labelled according to the Motor unit for which it has the highest Q-value after learning. The right map is coded according to the *Abstract* unit to which each Input unit belongs. This confirms that the abstraction process has learned to recognise the two distinct situations that the robot finds itself in — namely those requiring a left turn, and those requiring a right turn.

In this experiment it was assumed beforehand that there were only two abstract categories of interest, and that the Abstract network could efficiently utilise both units it was given. A second experiment uses an Abstract network with four units, the results of which are shown in figure F.18. The right plot shows how the four Abstract units were able to make a finer grained classification of the input space with each of the existing categories now represented by two Abstract units. In particular, it seems as if a distinction has been made between situations that involve just peripheral sensor activity, and those that do not (see figure F.19). This distinction was present in the majority of runs of this particular experiment indicating that there may be a significant



(a) The familiar Input and Motor maps plotted in input and motor space. Each Motor unit is shaded, as usual, according to its distance from the $\langle 0,0 \rangle$-$\langle 1,1 \rangle$ diagonal (see figure F.16), and each Input unit is shaded according to the Motor unit for which it maintains the highest Q-value.



(b) The Input map plotted as above, except that each Input unit is now shaded either white or black, depending on which of the two abstract categories it belongs. It is evident that the Abstract network has learned to dichotomise the input space into regions requiring a left turn, and regions requiring a right turn.

Figure F.15: The Input map after learning.

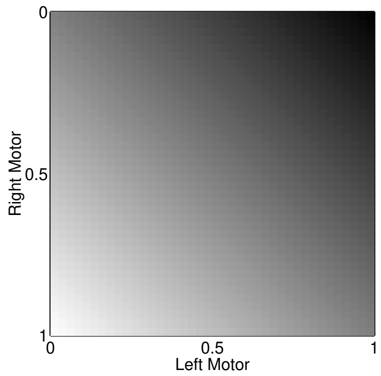


Figure F.16: Each Motor unit is shaded according to where it lies in the output space. Each Input unit can then be shaded the same as the Motor unit for which it maintains the highest Q-value.

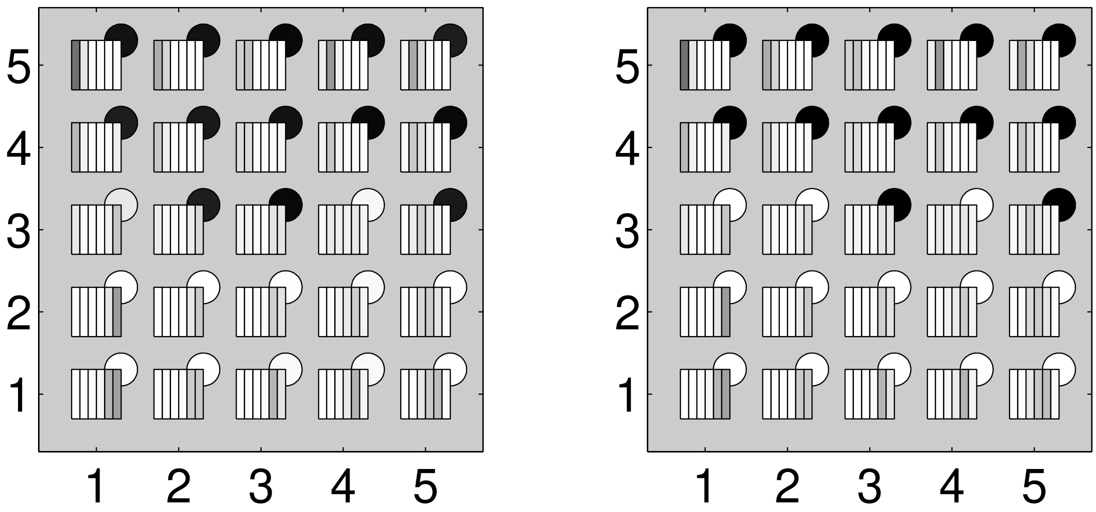


Figure F.17: The left plot shows the Input map in physical space, with each unit labelled with a circle shaded according to the Motor unit with the highest Q-value for that Input unit. The right plot also shows the Input map in the physical space of the network, but this time the units are shaded according to the *Abstract* unit to which they most closely belong.

difference between the expected returns of situations with just peripheral activity, and those with heavier frontal sensor activity.

However, it is noted that it is not always the case that the Abstract units distribute themselves evenly over the Input units. Illustrated in figure F.20 is another run of the same experiment, in which three Abstract units classify the situations requiring a right turn, leaving only a single unit for the situations requiring a left turn. This illustrates the importance of the Abstract network size, and in particular that there must be enough units to adequately represent every abstract category. This implies a number of units greater than the expected number of abstract categories, since some redundancy is inevitable. However, the cost of too many units is that the benefits of a more compact representation of the state space are compromised. It is also noted that an insufficient number of units will result in over generalisation, which was sometimes evident in this task when only two Abstract units were used. Four units proved a more reliable Abstract network size for this particular problem.

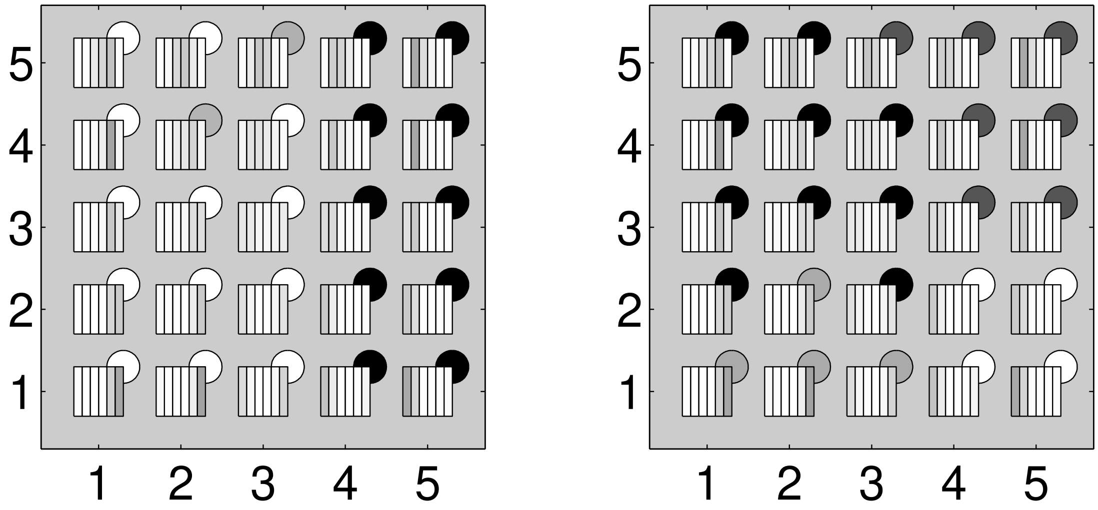


Figure F.18: The same experiment performed with four Abstract units which, in the right plot, are coloured black, white and two intermediate shades of grey. The abstraction process is now able to make a finer grained classification of the input space (see text). The map also illustrates how occasionally some Input units fail to learn either a sharp left or right turn as in the fourth and fifth rows here (see left plot). However, these units respond to situations with very little sensor activity of any kind, so their response is less likely to have a significant impact on the reward signal.

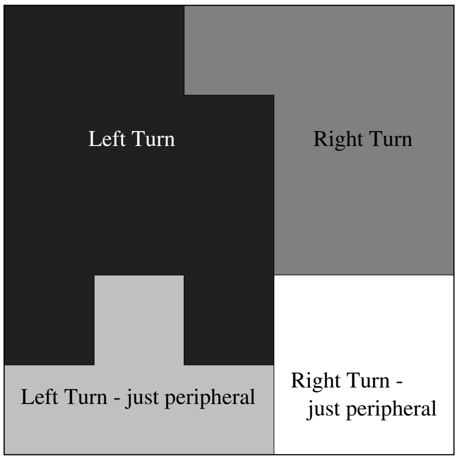


Figure F.19: An abstraction of the classification evident in figure F.18(right).

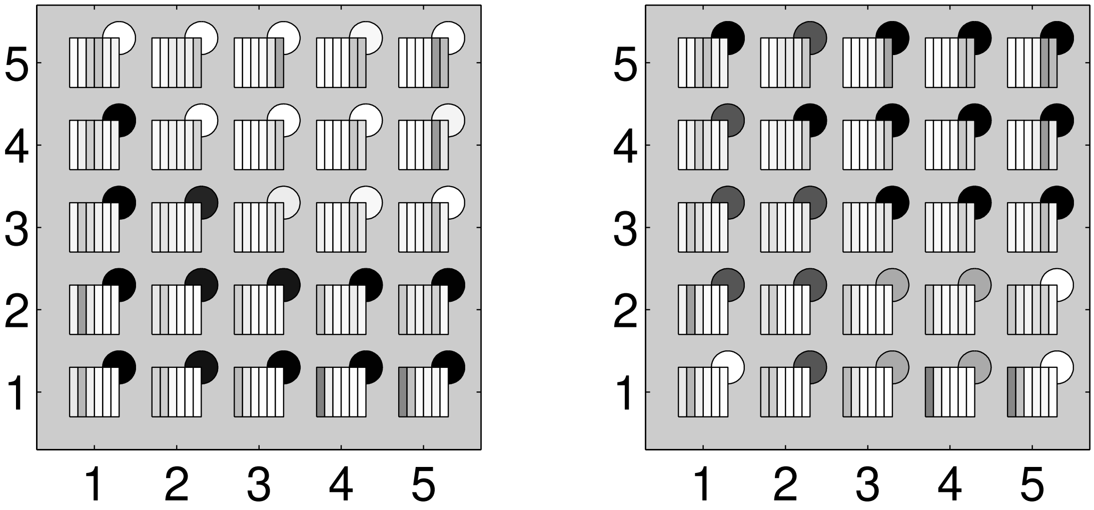


Figure F.20: Another run of the experiment involving four Abstract units. This time, rather than two units taking the 'left-turn' category and two units taking the 'right-turn' category, the network has split itself three to one. This highlights the lack of an equiprobable distribution guarantee and hence the difficulty in matching the number of Abstract units to the expected number of abstract categories.

F.2.5 Abstraction in the 'non-contiguous' mapping experiment

A final experiment in this section returns to the problem of mapping the function of figure F.21 which was introduced in section 5.6.7. Inputs, $\langle x, 0.5\rangle$, are generated with x randomly and evenly distributed in the range $[0,1]$, and the second constant of 0.5 used purely as an implementational convenience. The correct output is defined as $\langle x, 0\rangle$ if $0 \geq x < 0.1$ or $0.2 \geq x < 0.3$ or $0.4 \geq x < 0.5\ldots$ and $\langle x, 1\rangle$ otherwise, but these target actions are not known to the system beforehand. The system must discover a set of appropriate actions for itself, using only a scalar reward signal for feedback. Recall that the reward is the negative of the Euclidean distance (in output space) of the agent's response from the target action. Also recall that the reward is given immediately — i.e. the issue of delayed rewards does not arise here.

This was an important test for the basic learning algorithm because neighbouring points in the input space do not optimally map to neighbouring points in the output space. We might expect this to be a problem because of the neighbourhood updates involved in the learning rules, but in fact the results from this experiment (duplicated in figure F.22) suggested that as long as the neighbourhoods are annealed over the course of

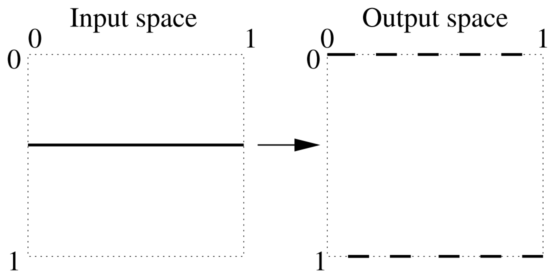
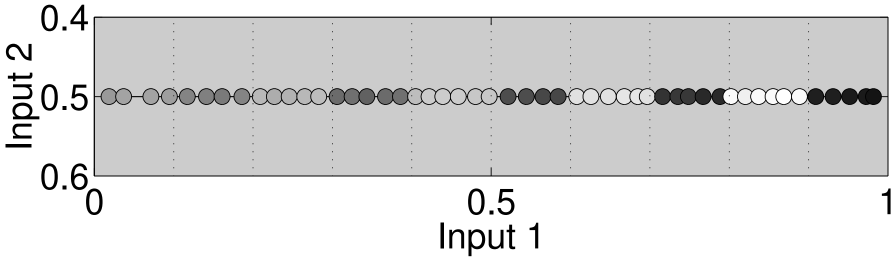
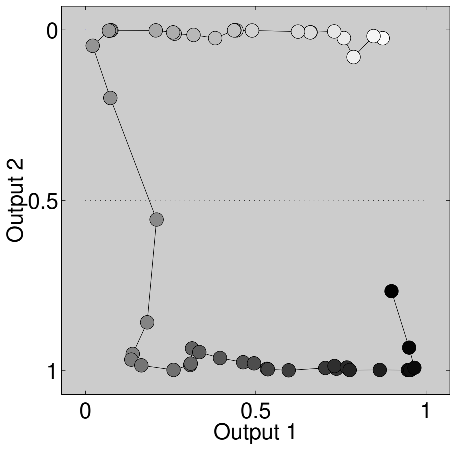


Figure F.21: Mapping to be learned. Duplicated from figure 5.41.



(a) Magnified plot of the Input map in the input space



(b) The Output map plotted in output space, shaded this time according to the units' index within the one-dimensional Output map.

Figure F.22: Results of the non-contiguous mapping experiment. The Input units are coded in the usual way, but the shading of the Output units is now based on topological index. Duplicated from figure 5.42.

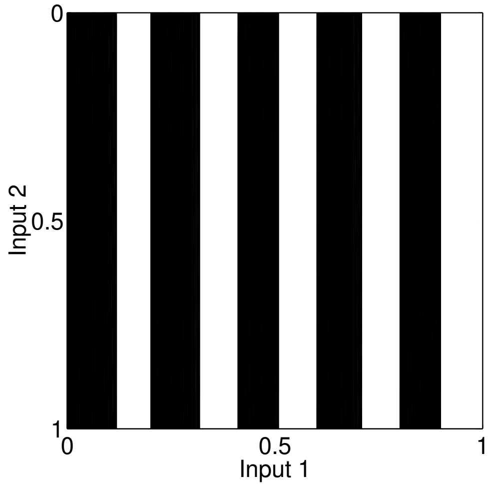


Figure F.23: Dichotomy of the input space. The black and white regions correspond to the categorisation performed by the Abstract network.

learning, this need not be the case.

It is now interesting to see how the abstraction process deals with this kind of problem. The first experiment involves an Abstract network with just two units. The system is effectively faced with the task of splitting the input space (represented by the Input units) into two classes, with each class behaving as consistently as possible with respect to how its regions behave in terms of the reward elicited under each of the currently available actions. Figure F.23 shows how the input space is split with black representing the regions closer to the prototype of the first Abstract unit, and white representing the regions classified by the second Abstract unit.

More interesting are the plots of membership against each point in the input space for each of the two categories, and these are shown in figure F.24. The plots confirm the result that one Abstract unit classifies 'strips' 1,3,5,7 and 9 and the other classifies strips 2,4,6,8 and 10. Obviously each class groups alternating strips because these require relatively similar actions. Each abstract class prototype sits over the middle of the space so that strips 4 and 5 are most unambiguously categorised. This is intuitive since the pull of the peripheral strips will balance each other out. Of interest are the small, local undulations that consistently appear at the centre of each strip (at the top of each peak). This is shown more clearly in the profile of the graphs in just the first, and relevant dimension of the input space (see figure F.25). The reason for these

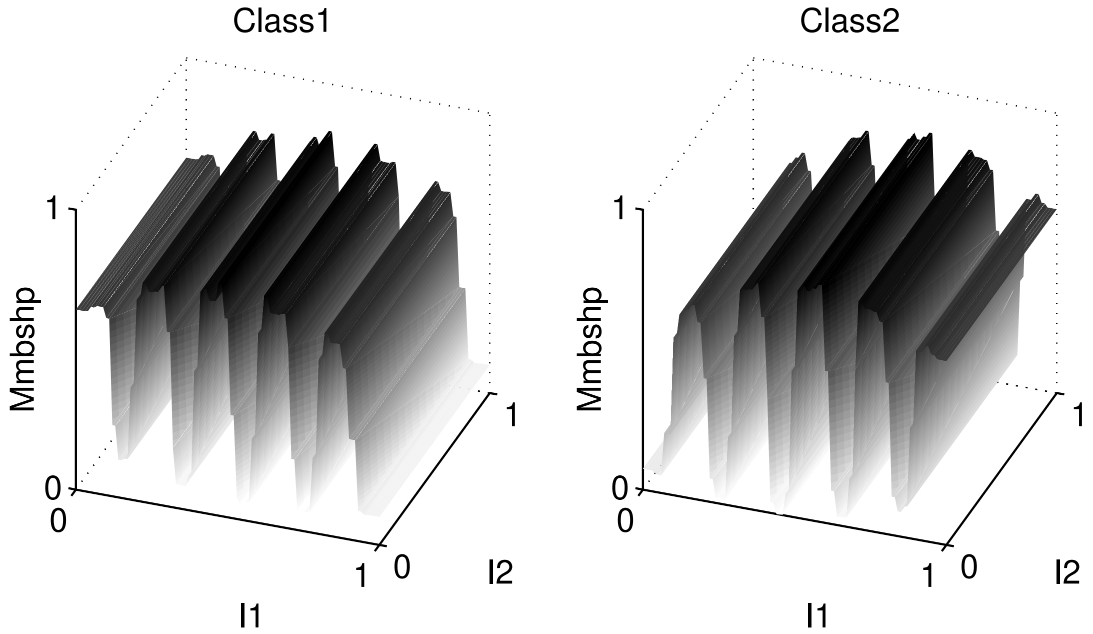


Figure F.24: Degree of membership of the input space to each of the two abstract classes.

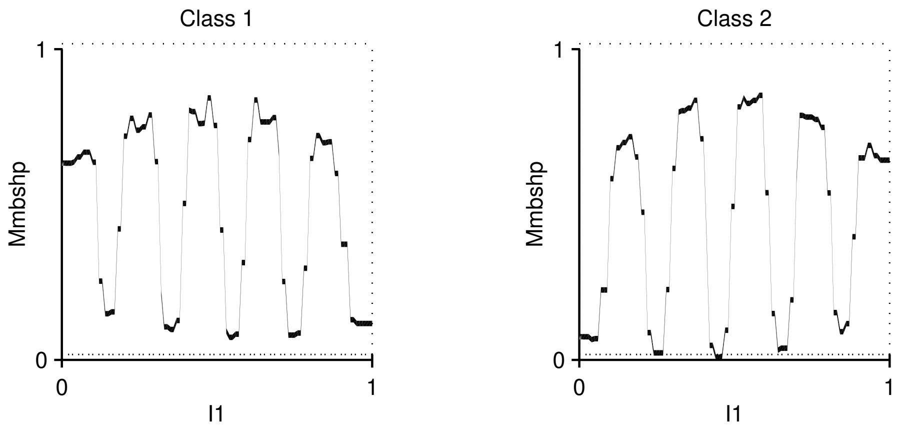


Figure F.25: Profile of figure F.24. The irrelevant second dimension of the input space is ignored. Note the perturbations at the centre of each peak.

perturbations is not clear at this point.

A second trial involves an Abstract network containing ten units. One hypothesis might be that each Abstract unit would learn to group all of the Input units within a single

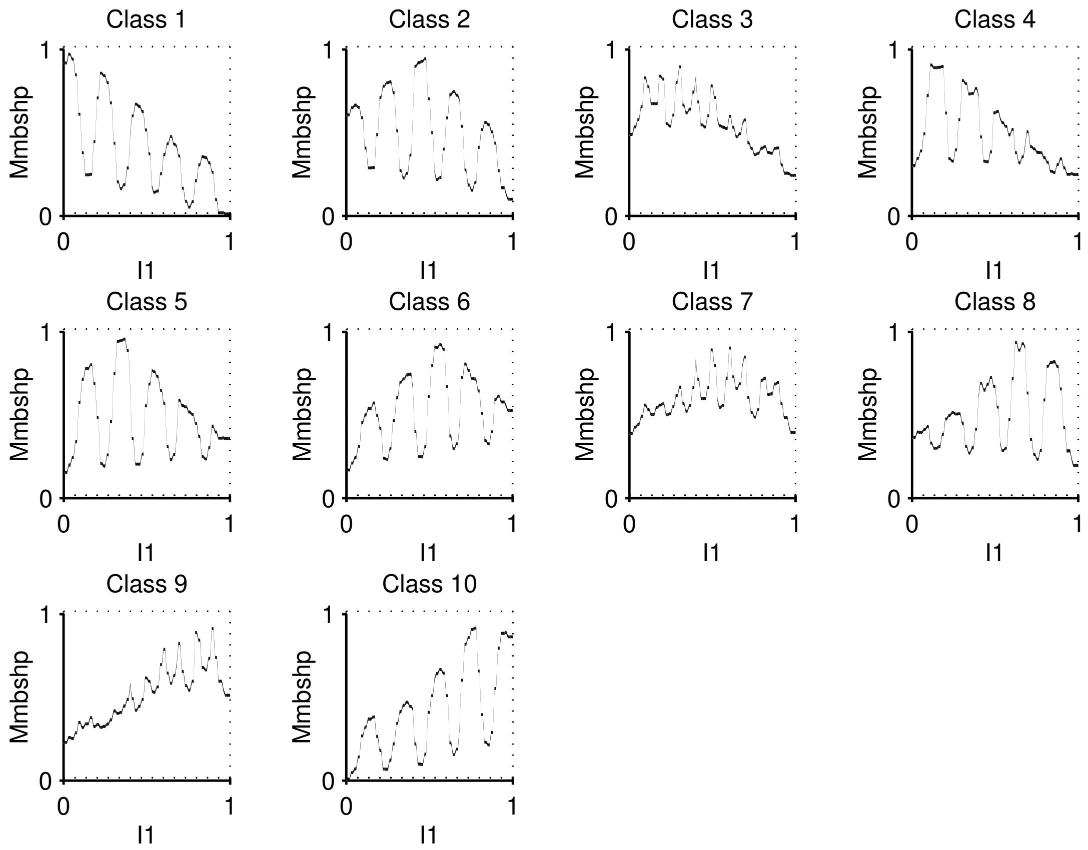


Figure F.26: Profile of membership plotted against the first input, I1 for each of ten abstract categories.

column (i.e. $x = 0$ to 0.1 or $x = 0.4$ to $x = 0.5$ etc.), but the network does not succeed in doing this. As figure F.26 shows, class one preferentially responds to strip 1 with decaying classification of strips 3,5,7 and 9. The second class identifies strip 5 most clearly ($I1 = 0.4$ to $I1 = 0.5$), but also 3 and 7 strongly. Class four switches to the even strips, particularly strips 2 and 4, and so on.

The upshot is that each of the ten strips is represented by one of seven abstract categories, with some strips having to share categories. The reason for this is that the Abstract map clearly has folds in the topology which are evident at the points where the map switches from classifying the even to the odd strips. At each of these points — namely classes three, seven and nine — these classes suffer interference from their neighbours in the network. We see again an example of the effect of network units being strung out between clusters as they are pulled in opposite directions with equal force. Ameliorating this problem boils down to the need to achieve unfolded maps which preserve as much of the topology of the data as possible. In this particular instance, the prefered solution would presumably be to have class one prototypically classify strip 1, class two classify strip 3, class three strip 5 etc., and then have the second half of the map classify the even strips, again in order. Only then would the network have a chance to categorise each strip of reward-consistent input space with a distinct Abstract unit.

These 'interference' categories may explain the perturbations in the peaks of the two membership plots of figure F.25. Since a minimum neighbourhood size of 1 was used for the Abstract network, there was always a pull on each of the two Abstract units by the other unit. Since the membership peaks of one unit overlap with the troughs of the other, this will tend to cause interference, with the effects most strongly felt at the extremes.

F.2.6 Discussion

Since the discovery of abstract classes has been reduced to a clustering problem, there are clearly a number of alternatives to using a SOM. However, the discussion follows very similar lines to that of chapter 6, and so is not duplicated here. However, there are a few simple variations to the basic algorithm presented here which are worth considering briefly.

F.2.6.1 On-line vs off-line learning

Currently, the Q-vectors are presented to the Abstract network in the order that the Input units become active. Since it is already known that the Input map does not maintain an equiprobable distribution, it is expected that some Input units will be classified less diligently than others, with the possibility that some Input units actually never submit their Q-vector for abstract categorisation and are thus categorised only through generalisation. One alternative is to present the Q-vector of each Input unit to the Abstract network with equal probability, irrespective of when and indeed whether or not that

Input unit is active. If the update is performed randomly (or sequentially) in this way, then learning becomes an off-line, non-interactive process that is independent of unit usage.

F.2.6.2 Winner selection

There may be other suitable winner selection methods apart from the Euclidean distance measure used here. In experiments not reported here, using the cosine of the angle between input and weight vectors as a distance measure was found to yield slightly different abstract categories. The issue is not investigated further, but it is noted that the concept of similarity between Q-vectors is open to interpretation.

F.2.6.3 Similarity definitions

Instead of defining two Input units as belonging to the same class if and only if they elicit the same reward under all actions, one alternative is to invoke the weaker definition that for two units to be considered the same, they must propose similar optimal actions in terms of distance in output space. How units behave (in terms of reward) under sub-optimal actions is now irrelevant. This could be an advantage because as exploration of sub-optimal actions is decreased during a trial, the Q-values associated with those actions are likely to drift as they receive fewer and fewer direct updates as a result of first hand experience. Not considering irrelevant actions in the abstract categorisation of Input units simplifies the classification procedure which now becomes a clustering problem in the output space rather than Q-matrix space. However, the algorithm pays by losing some of its subtlety. For example, it would no longer be possible to discover the 'diagonal' abstract category of figure F.9. A key question is "What does it mean for two Input units to be similar?". This question is also clearly open to interpretation.

F.3 Abstracting over the Output map

Consider again a simplified system with an Input map containing only three units and an Output map containing just four. At the beginning of this chapter, the Q-values of this system were written down as a 3x4 matrix:

$$\begin{bmatrix} a1 & a2 & a3 & a4 \\ b1 & b2 & b3 & b4 \\ c1 & c2 & c3 & c4 \end{bmatrix}$$

with element (m,n) representing the Q-value between the mth Input unit and the nth Output unit. It was noted that the *Q-vector*:

$$\begin{bmatrix} a1 & a2 & a3 & a4 \end{bmatrix}$$

represents the Q-values from the first Input unit to each of the four Output units, and can therefore be interpreted as characterising that first unit in terms of its reward profile. Discovering classes of similar Input units was then reduced to a clustering problem in this Q-vector space.

This solution was based on the following definition:

Two Input units (S and T) are similar if and only if they yield similar Q-values for every action. i.e. $Q(S,a) \approx Q(T,a)$***, for all a.***

It now seems reasonable that this assumption can be adapted to provide a basis for clustering *Output* units. The justification for wanting to do this is the same as for the input space. Consider how the Motor units tended to cluster in one of two corners when learning to avoid obstacles in the Khepera simulator. A more compact representation could represent this space with only two units, and this has natural implications for more efficient re-use and adaptation of the Motor map. So a second definition is given:

Two Output units (U and V) are similar if and only if all states yield similar Q-values for these two actions. i.e. $Q(s,U) \approx Q(s,V)$***, for all s.***

Now the problem of abstracting over the output space is reduced to one of clustering the *columns* of the Q-matrix [3], so that Output unit one is considered similar to Output unit two if and only if:

$$\begin{bmatrix} a1 \\ b1 \\ c1 \end{bmatrix} \approx \begin{bmatrix} a2 \\ b2 \\ c2 \end{bmatrix}$$

in the Euclidean sense. The obvious approach is to use yet another Kohonen map to cluster these Q-columns and the augmented architecture can now be visualised along the lines of figure F.27.

As a further illustration, consider figure F.28. This is a duplication of figure 5.42 and shows the strength of every connection between the Input and Output maps for a particular solution of the 'non-contiguous' problem of section 5.6.7. This effectively represents the Q-matrix for that solution. Abstraction over the Input network is achieved by clustering the columns of this plot, and abstraction over the Output network achieved by clustering the rows.

The algorithm, parameters and discussion are similar to the abstract learning of the Input map, so some results of abstracting over the Output space are presented directly. First, figure F.29 shows the state of the Motor map after learning the obstacle avoidance task in the Khepera simulator. Two Abstract Motor units are used, and these are represented by the two colours (black and white) in the second plot. It is evident that the abstraction process discovers two categories of actions — left turns, and right turns. It is important to remember that the clustering is not taking place in the output space as it may appear from the diagram, but in the Q-column space of the Q-matrix. Hence there is no suggestion that neighbouring units in the output space should necessarily belong to the same abstract category. However, in this case they do because Output units that propose similar actions will tend to yield similar rewards given the nature of the task. Note again the stringing effect of the Motor map, with two units caught between the two clusters.

[3] Let us call these *Q-columns*.

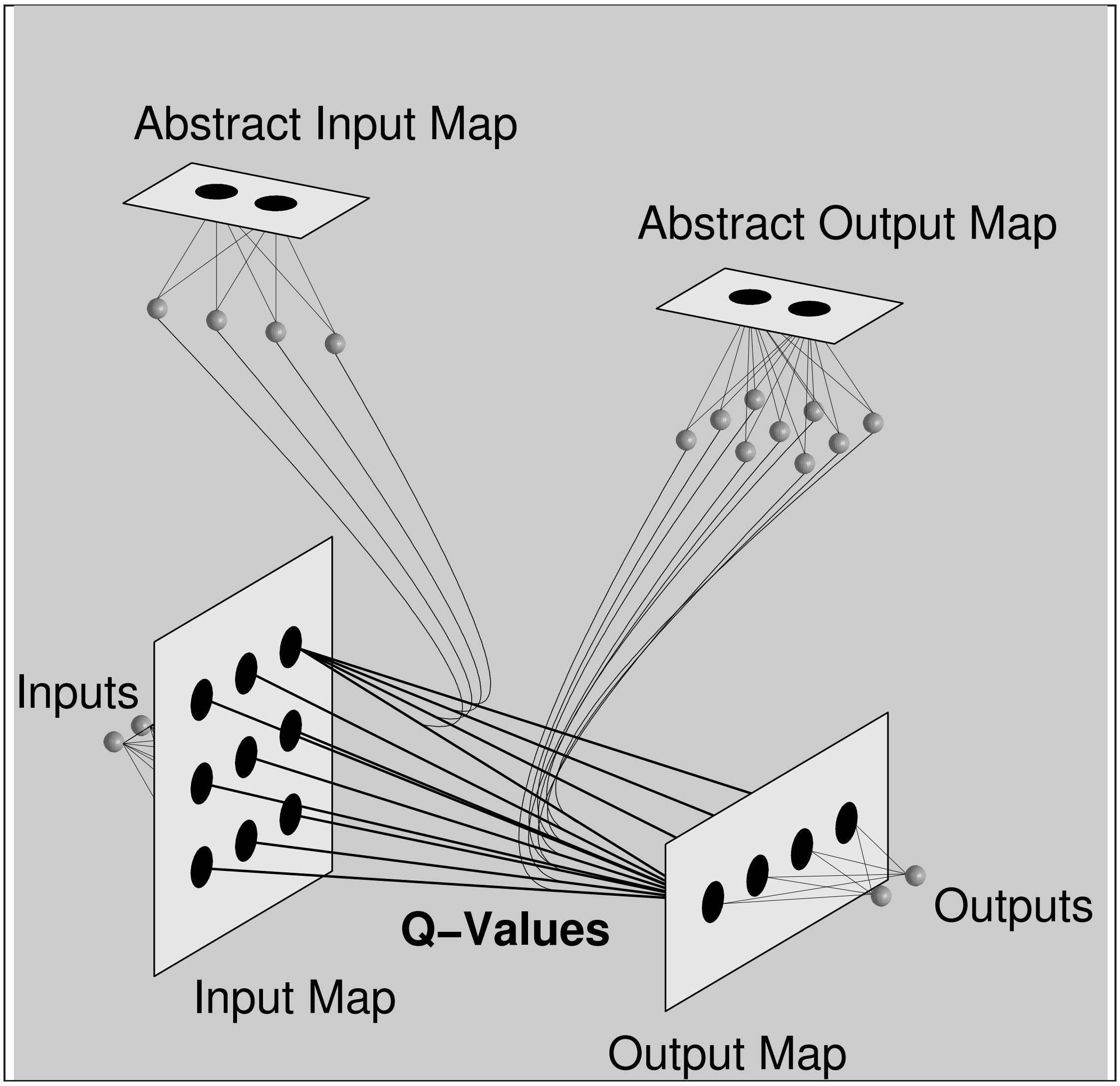


Figure F.27: The full architecture. Now an *Abstract Output map* is added that clusters the vectors of Q-values from each Output unit to every Input unit. Again, not all connections are shown.

The degree of membership of each point in the output space to each of the two categories is illustrated in figure F.30. The graphs look rather unexpected in that the classification seems to have taken place along one axis. The reason for this is that only a small region of the output space (the two corners) is occupied by the Output map. Attempting to classify the unmapped regions of the output space is meaningless in this example. A more useful diagram is that of figure F.31 which shows the degree of membership of each Output unit to each category. From this diagram it is confirmed

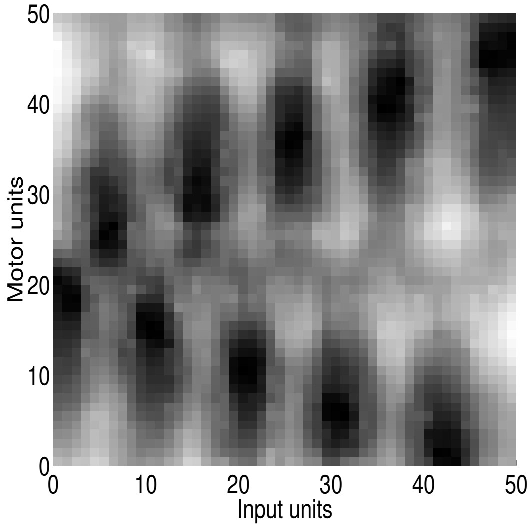
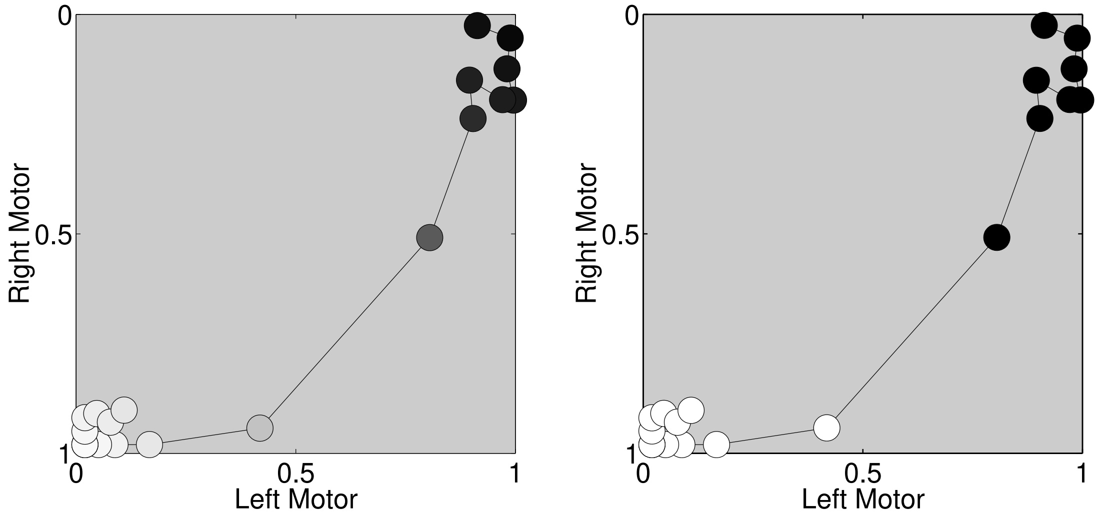


Figure F.28: A graphical illustration of the Q-table (or Q-matrix) after learning in the 'non-contiguous' experiment of section 5.6.7. This figure is duplicated from figure 5.42. Abstraction over the input network can be achieved by clustering the columns of this table and abstraction over the Output network can be achieved by clustering the rows.



(a) The Motor network plotted in output space. Units are coloured in the usual way according to their position in the output space (see figure F.16)

(b) The same Motor network is coloured according to which of two abstract motor categories each unit belongs.

Figure F.29: The Output map after learning the obstacle avoidance task.

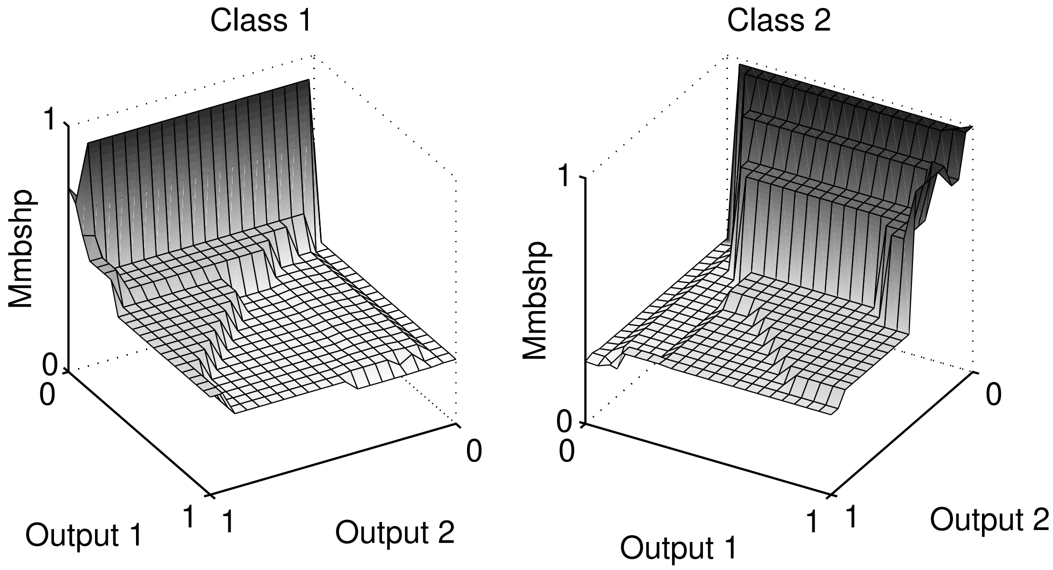


Figure F.30: A plot of abstract category membership against the two dimensions of the output space. Calculating the membership of each point is a two stage process. First a point is chosen in output space and the nearest Output unit is selected as the unit most faithfully representing this point. Next, the Euclidean distance between the Q-column of that Output unit and each Abstract unit is calculated. These distances are then normalised (by taking the inverse and then scaling to the range $[0, 1]$).

that class one categorises the left turns, and class two the right turns. Output units eight and nine are the two units which are strung out between the two clusters. Membership to both Abstract categories is low at this point since neither unit satisfies the prototype of either Abstract unit.

It is again interesting to see what happens when extra units are added to the Abstract network. In a similar experiment, figure F.32 shows the distribution of Motor units in output space labelled according to which of *three* abstract classes those units belong. The third category classifies the strung-out units that lie between the two well defined clusters. Precise membership to each of the three categories is illustrated in figure F.33. That the three strung-out units form their own class is not surprising, since these actions are very different from the extremes of turning left and turning right on the spot, and will in general result in lower reward in most situations.

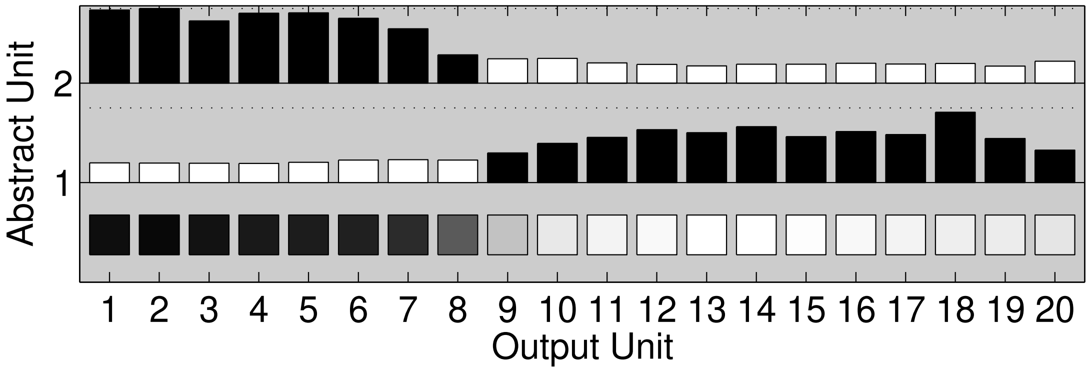


Figure F.31: The membership of each of the twenty Output units to each of the two Abstract Output categories. Each Output unit is plotted along the horizontal axis — shaded for convenience according to the usual convention of figure F.16 (see also F.29(a)). Membership of each unit to each category is indicated by the height of the bars above each unit. For each Output unit, the bar of one of the Abstract units is shaded black to signify that this is the abstract class to which that Output unit belongs. Hence the black bar is always the highest in its column. This kind of plot has the advantage of showing abstract categorisation for only the mapped parts of the output space.

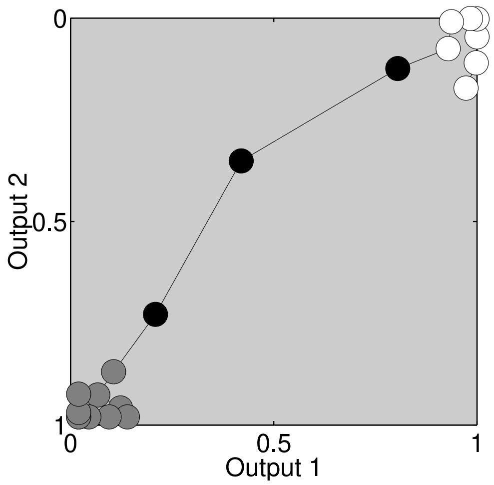


Figure F.32: Output map after learning the obstacle avoidance task with three Abstract Output units. Each unit is shaded according to which of the three abstract units it belongs.

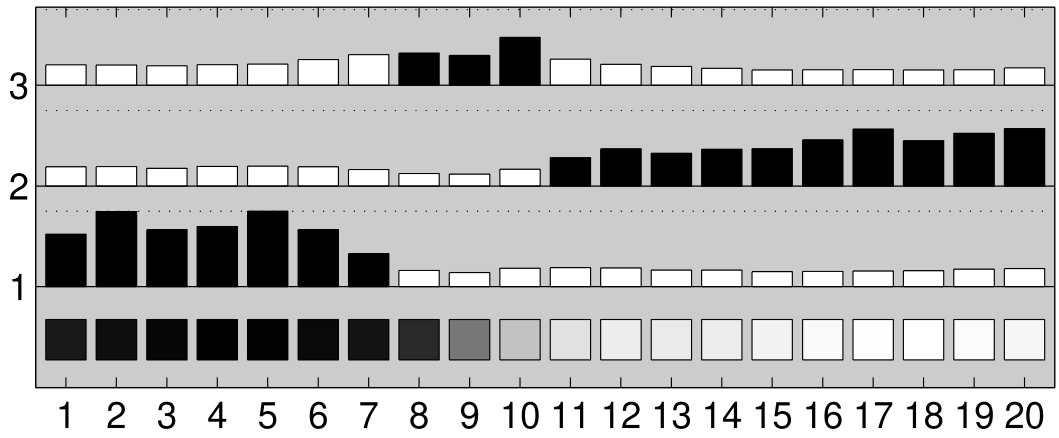


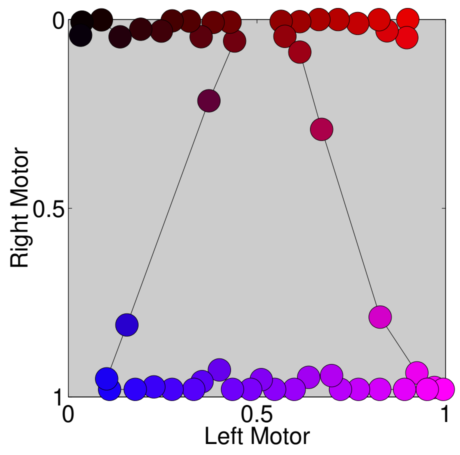
Figure F.33: Normalised membership of each Output unit to each of the three abstract classes for the obstacle avoidance problem.

F.3.1 The non-contiguous task

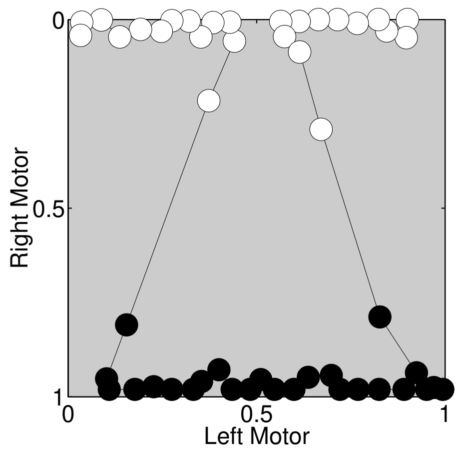
The process of abstracting over the output space is now tested on the non-contiguous problem of figure F.21. Again, the details of the experiment are familiar, so the results are presented directly in figures F.34 to F.36. The Abstract map again clusters the Output units as if the clustering was taking place in the output space because Output units close in output space tend to have similar Q-columns[4]. The effect of using more Abstract units is not shown here, but the predictable result is that the extra units are employed in subdividing the existing classes approximately equally.

Much of the discussion pertaining to the abstract mapping of the output space is identical to that of the input space, and is therefore not duplicated here. In particular, parameter settings, the effects of different numbers of Abstract units, the possibility of using different clustering algorithms, the distinctions between on-line and off-line updates, and alternative winner selection criteria, are equally relevant here.

[4]This is a natural consequence of the formulation of the reward signal in which actions are rewarded proportionally to their distance in output space from the desired action.



(a) The Motor network plotted in output space. Units are coloured according to their position in the output space (see figure F.35)



(b) The same Motor network is coloured according to which of two abstract motor categories each unit belongs.

Figure F.34: The Output map after learning the non-contiguous task of figure F.21.

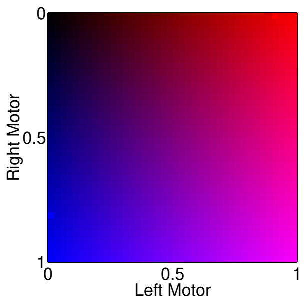


Figure F.35: Each Motor unit is colour coded according to where it lies in the output space. The horizontal position dictates the amount of red at that point, and the vertical position the amount of blue. This approach has the advantage of giving every point in the two-dimensional space a unique label.

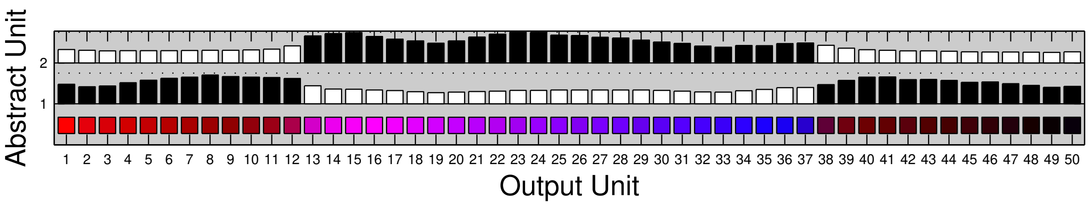


Figure F.36: Normalised membership of each Output unit to each of the two abstract classes for the non-contiguous task of figure F.21.

F.4 Applications

Having derived a small number of abstract classes, we now consider how this information could be exploited. An obvious argument is that if both Input and Output Abstract units can be re-used in learning further, related tasks, then these tasks could benefit from the speed up factor offered by a more compact representation. A second advantage is that the generalisation offered by the Abstract network begins to provide a primitive set of symbols and rules for behaviour. The problem with working with user-defined symbols is that they may not turn out to be the right symbols for the task. The problem with not using symbols, and relying solely on grounded sub-symbolic representations, is that the system often operates as a black box with no ability to ask "Why?" or make sense of and learn from the generalisations made by the system. If we can embrace the abstractive power of symbols, while ensuring that these symbols are grounded through the sub-symbolic interactions of the system itself with its environment, then the best of both worlds may be achieved. This is part of the motivation behind this section on abstract categories.

As an example, consider the obstacle avoidance problem, and a specific sensory stimulus invoking a motor response. In a crude way, it is now possible to ask why that particular response was elicited from the environment. The Abstract Input unit to which the stimulus belongs can be identified, and then the system run in reverse so that all the regions of the input space categorised by that Abstract unit are activated. Now all the sensory stimuli corresponding to either 'more left sensor activity than right' or 'more right than left' can be offered as a crude explanation for the action. Of course it requires further abstraction before the highlighted space could actually be given a label like 'more left than right', but there is now at least some meaning in the representation that emerges out of regularities in the reward signal.

It may also be possible to exploit the formation of abstract categories for expediting the acquisition of behaviours. It has already been noted in section F.2.3 that regularities in the Input map can be identified by the Abstract Input network before either the Q-table or the Motor map are in their final states. Q-learning could now be performed between the Abstract Input and Abstract Output maps simultaneously with the Q-learning between the basic Input and Output maps. The vastly smaller number of state-action pairs to be learned in the former case may enable learning to proceed faster. One difficulty however, is that to utilise the Abstract maps in this way, they have to be given autonomy of the agent and over the maps underneath. But the lower maps are still forming, and more importantly still contributing to the formation of the abstract categories above, and so *they* still require autonomy at this point. If a suitable schedule for switching from the low level maps to the Abstract maps can be found however, there may be an advantage to be gained by this representational re-description (to hijack the terminology of Karmiloff-Smith (1995)).

Another way to use the Abstract maps to speed up learning might be to propagate Q-updates to all the states and actions in the abstract classes of the current state-action pair. In this way states (and actions) learn from each other if they have behaved similarly in the past with respect to the reward signal, and hence a more intelligent concept of neighbourhood is introduced.

It is also natural to consider what happens when this abstraction process is extended. For example, following the previous discussion, it is easy to conceive of an architecture along the lines of figure F.37 with higher order Abstract maps, each containing fewer units than the one below. This addresses the issue of finding an appropriate number of Abstract units and removes the need for adding and removing units in a constructive fashion. Early Abstract maps containing more units would tend to under-generalise, while later Abstract maps containing fewer units would tend to over-generalise. Since over-generalisation would tend to manifest itself in high Q-vector variance within each category, it should be possible for the system to choose the compactest representation which does not compromise the resolution requirements of the task in hand. However, this could also be achieved with a flat model — i.e. with a number of Abstract maps of different sizes abstracting over the same low level Input map. The benefits of a hierarchical system only materialise when there is processing performed at each level that is dependent on the abstractions made at that level.

F.5 Summary

Karmiloff-Smith discusses child development in her book *Beyond Modularity* with an emphasis on the iterative, incremental and interactive aspect to forming representations of the world. Her thesis could be paraphrased as advocating a process of abstraction (or in her more general words *redescription*) which takes place on a child's current representations of a problem. The new representations that are formed as a result of this redescription can then be used to fuel a new round of environmental interaction that can be both more efficient and more effective. This new round of interaction can then generate more salient and appropriate symbols and so on. She provides evidence for the redescription occurring in stages with development a step-wise process and the transition from one set of representations to the next set only occurring when sufficient confidence is held in the status of the new representations. This supports the hypothesis that these representational changes are qualitative, and the product of something more than just a slow, continuous and homogeneous adaptive process.

It would be interesting to model this kind of psychological theory, perhaps within the context of RL. Karmiloff-Smith's ideas fit in well with the aspirations of designers of learning agents and RL researchers to whom finding appropriate symbols to represent the world is a key issue. This chapter only briefly considers the idea of grounded symbols, but preliminary results suggest that further research in this direction may be profitable.

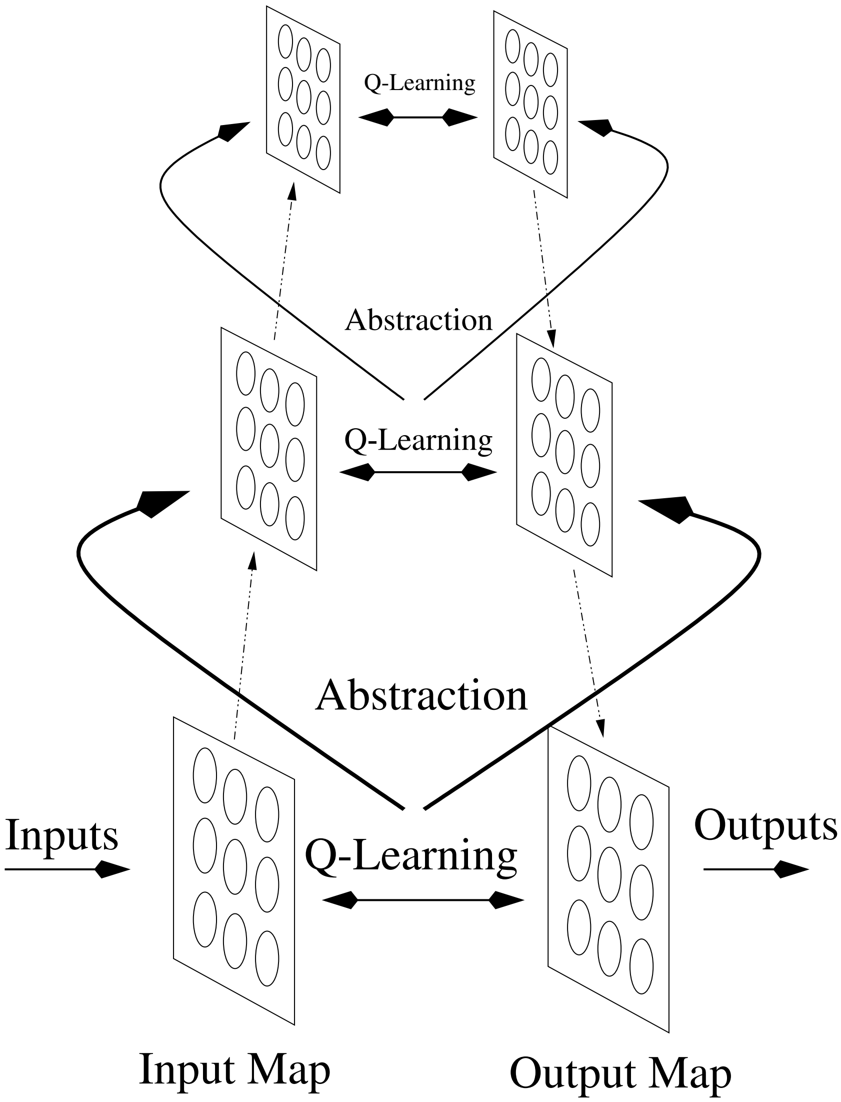


Figure F.37: A hypothetical continuation of the generalisation process with learning proceeding in parallel at each level of the hierarchy. An increasing compaction of the representation is achieved. The dotted arrows are intended to show the flow of control through the system with classification of the input stimulus taking place as high up the hierarchy as possible, and control being fed down to the low level Output units.

Bibliography

Ackley, D. H. (1989). Associative learning via inhibitory search. In Touretzky, D., editor, *Advances in Neural Information Processing Systems*, volume 1, pages 20–28. Morgan Kaufmann, San Mateo, CA.

Ackley, D. H. and Littman, M. L. (1990). Generalisation and scaling in reinforcement learning. In Touretzky, D., editor, *Advances in Neural Information Processing Systems*, pages 550–557. Morgan Kaufmann, San Mateo, CA.

Albus, J. (1975). A new approach to manipulator control: The cerebellar model articulation controller (cmac). *Dynamic Systems, Measurement and Control*, 97(3):220–227.

Albus, J. (1981). *Brains, Behaviour, and Robotics*. BYTE publications, Peterborough, N.H.

Anderson, C. W. (1986). *Learning and Problem Solving with Multi-layer Connectionist Systems*. PhD thesis, University of Massachusetts.

Angeniol, B., de la Croix Vaubois, G., and Texier, J. L. (1988). Self-organising feature maps and the travelling salesman problem. *Neural Networks*, 1:289–293.

Araujo, E. G. and Grupen, R. A. (1996). Learning control composition in a complex environment. In Maes, P., Mataric, M. J., Meyer, J.-A., Pollack, J., and Wilson, S. W., editors, *From Animals to Animats 4. Proceedings of the fourth international* *conference on simulated and adaptive behavior*, pages 333–342, Cambridge, Massachesetts. London, England. MIT Press.

Arkin, R. C. (1986). Path planning for a vision based autonomous robot. In *Proceedings of the SPIE Conference on Mobile Robots*, pages 240–249, Cambridge, Massachusetts.

Arkin, R. C. (1991). *Behaviour Based Robotics*. MIT Press, Cambridge, Massachusetts.

Baldi, P. and Hornik, K. (1989). Neural networks and principal component analysis: learning from examples without local minima. *Neural Networks*, 2(1):53–58.

Barnsley, M. (1988). *Fractals everywhere*. Academic Press.

Barto, A. G. and Anandan, P. (1985). Pattern recognising stochastic learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 15:360–374.

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

Bechtel, W. and Abrahamsen, A. (1991). *Connectionism and the Mind*. Blackwell, Cambridge, Massachusetts.

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

Bishop, C. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press.

Bishop, C., Svensen, M., and Williams, C. (1998). Developments of the generative topographic mapping. *Neurocomputing*, 21:203–224.

Blum, E. K. and Li, L. K. (1991). Approximation theory and feedforward networks. *Neural Networks*, 4(4):511–515.

Bourlard, H. and Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59:291–294.

Braitenberg, V. (1984). *Vehicles*. MIT Press, Cambridge, Massachusetts.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2, No1:14–23.

Brooks, R. A. (1990). Elephants don't play chess. In Maes, P., editor, *Designing Autonomous Agents*, pages 3–15. MIT Press.

Brooks, R. A. (1991a). Intelligence without reason. Technical Report A.I.Memo No 1227, MIT Artificial Intelligence Laboratory.

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

Bryson, J. (1996). The design of learning for an artifact. Technical report, Intelligent Systems Laboratory, Department of Psychology, Edinburgh University.

Carpenter, G. A. and Grossberg, S. (1987a). Art 2:self-organisation of stable category recognition codes for analog input patterns. *Applied Optics*, 26:4919–4930.

Carpenter, G. A. and Grossberg, S. (1987b). A massively parallel architecture for a self-organsing neural pattern recognition machine. *Computer Vision: Graphics and Image Processing*, 37:54–115.

Chapman, D. and Kaelbling, L. P. (1991). Input generalisation in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Connell, J. (1992). ”sss: A hybrid architecture applied to robot navigation”. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2719–2724, Nice, France.

Crites, R. H. and Barto, A. G. (1996). Improving elevator performance using reinforcement learning. *Neural Information Processing Systems*, 8.

Dawkins, R. (1982). *The Extended Phenotype*. Oxford University Press.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *journal of Royal Statistical Society*, 39(1):1–38.

Digney, B. (1996). Emergent hierarchical control structures: Learning reactive/hierarchical relationships in reinforcement environments. In Maes, P., Mataric, M. J., Meyer, J.-A., Pollack, J., and Wilson, S. W., editors, *From Animals to Animats 4. Proceedings of the fourth international conference on simulated and adaptive behavior*, pages 363–372, Cambridge, Massachusetts. London, England. MIT Press.

Dorigo, M. (1995). Alecsys and the autonomouse: Learning to control a real robot by distributed classifier systems. *Machine Learning*, 19:209–240.

Dorigo, M. and Colombetti, M. (1994). Robot shaping: Developing autonomous agents through learning. *Artificial Intelligence*, 71(2):321–370.

Durbin, R. and Willshaw, D. (1987). An analogue approach to the travelling salesman problem using an elastic net method. *Nature*, 326:689–691.

Edelman, G. (1987). *Neural Darwinism: The Theory of Neuronal Group Selection*. Basic Books.

Edelman, G. and Finkel, L. H. (1990). Neuronal group selection in the cerebral cortex: Dynamic aspects of neocortical function. In J. A. Anderson, A. P. and Rosenfeld, E., editors, *Neuro-Computing 2*, pages 308–334.

Erwin, E., Obermayer, K., and Schulten, K. (1992). Self-organising maps: Ordering, convergence properties and energy functions. *Biological Cybernetics*, 67:47–55.

Fahlman, S. (1989). *Faster learning variations on Back-Propagation: An empirical study*, pages 38–51. Morgan Kaufmann, San Mateo.

Favata, F. and Walker, R. (1991). A study of the application of kohonen-type neural networks to the travelling salesman problem. *Biological Cybernetics*, 64:463–468.

Fritzke, B. (1995). A growing neural gas network learns topologies. *Advances in Neural Information Processing Systems*, 7:625–632.

Gallistel, C., Brown, A. L., Carey, S., Gelman, R., and Keil, F. C. (1991). Lessons from animal learning for the study of cognitive development. In Carey, S. and Gelman, R., editors, *The Epigenesis of Mind*, pages 3–36. Lawrence Erlbaum, Hillsdale, NJ.

Georgeff, M. and Lansky, A. (1987). Reactive reasoning and planning. In *Proceedings of AAAI-87*, pages 677–682.

Gillies, A. (2001). Personal communication. Institute for Adaptive and Neural Computation, Edinburgh, U.K.

Goodhill, G. (1992). *Correlations, Competition, and Optimality: Modelling the Development of Topography and Ocular Dominance*. PhD thesis, University of Sussex at Brighton.

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

Hartman, E. J., Keeler, J. D., and Kowalski, J. M. (1990). Layered neural networks with gaussian hidden units as universal approximations. *Neural Computation*, 2:210–215.

Hayes, G. and Demiris, J. (1994). A robot controller using learning by imitation. In *Proceedings of the 2nd International Symposium on Intelligent Robotic Systems, Grenoble, France.*, pages 198–204.

Hertz, Krogh, and Palmer (1994). *Introduction to the theory of neural computation*. Addison-Wesley.

Heskes, T. (1996). Transition times in self-organising maps. *Biological Cybernetics*, 75:49–57.

Heskes, T. (1999). *Energy functions for self-organising maps*, pages 303–315. Elsevier.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences*, volume 79, pages 2554–2558.

Hulle, M. M. V. (2000). *Faithful Representations and Topographic Maps*. J. Wiley and Sons, Inc.

Jaakkola, T., Jordan, M., and Singh, S. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6:1185–1201.

Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1(4):295–307.

Jones, L. K. (1990). Constructive approximations for neural networks by sigmoidal functions. *Proceedings of IEEE*, 78(10):1586–1589.

Kaelbling, L., Littman, M., and Moore, A. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence*, 4:237–285.

Kangas, J. and Kaski, S. (1998). 3043 works that have been based on the self-organising map (som) method developed by kohonen. Technical report, Helsinki University of Technology, Laboratory of Computer Science.

Karmiloff-Smith, A. (1995). *Beyond Modularity*. MIT Press.

Kohonen, T. (1987). *Self Organisation and Associative Memory*. Springer-Verlag, Berlin, 2 edition.

Kohonen, T. (1995). *Self Organising Maps*. Springer-Verlag, Berlin.

Kolmogorov, A. N. (1957). On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition. *Doklady Akademiia Nauk SSSR*, 114:953–956.

Kube, R. C. and Zhang, H. (1994). Collective robotics: From social insects to robots. *Adaptive Behavior*, 2(2):189–218.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In Touretzky, D., editor, *Neural Information Processing Systems*, volume 2. Morgan Kaufman.

Leow, W. K. (1998). Computational studies of exploration by smell. *Adaptive Behavior*, 6(3/4):411–434.

Li, G. (1999). *Towards On-Line Learning Agent for Autonomous Navigation*. PhD thesis, Chalmers University of Technology, Goteborg, Sweden.

Li, G. and Svensson, B. (1996). A multiple neural network based approach for reactive robot navigation. In *Proceedings of the fifth European workshop on Learning Robots*, pages 53–62.

Li, G. and Svensson, B. (1999). Navigating with a focus-directed mapping network. *Autonomous Robots*, 7(1):9–30.

Lin, L.-J. (1991). Programming robots using reinforcement learning and teaching. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 781–786, Cambridge, Massachusetts.

Lin, L. J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3-4):293–322.

Lin, L. J. (1993). *Reinforcement Learning for Robots using Neural Networks*. PhD thesis, Carnegie Mellon University, Pittsburgh. CMU-CS-93-103.

Lloyd, S. P. (1982). Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137.

Lyons, D. and Hendriks, A. (1995). Planning as incremental adaption of a reactive system. *Robotics and Autonomous Systems*, 14, No. 4:255–288.

Maes, P. and Brooks, R. (1990). Learning to coordinate behaviours. In *Proceedings of the Eighth International Conference on Artificial Intelligence (AAAI 90)*, pages 796–802, Bostson, MA.

Mahadevan, S. and Connell, J. (1991). Automatic programming of behaviour-based robots using reinforcement learning. In *Proceedings of the Ninth International Conference on Artificial Intelligence (AAAI 91)*, pages 768–773, Anaheim, CA.

Mataric, M. J. (1994). Reward functions for accelerated learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 181–189, Cambridge, Massachusetts.

Mataric, M. J. (1997). Reinforcement learning in the multi robot domain. *Autonomous Robots*, 4(1):73–83.

McCulloch, W. and Pitts, W. (1947). How we know the universals: The perception of auditory and visual forms. *Bulletin of Mathematical Biophysics*, 9:127–147.

Michel, O. (1996). *Khepera Simulator version 2.0. User Manual*. Originally downloaded from http://www.i3s.unice.fr/.

Mignault, A. and Marley, A. A. J. (1997). A real time neuronal model of classical conditioning. *Adaptive Behavior*, 6(1):3–61.

Minsky, M. (1986). *The Society of Mind*. Simon and Schuster, New York.

Minsky, M. and Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press.

Mondada, F., Franzi, E., and Ienne, P. (1993). Mobile robot miniaturisation: A tool for investigation in control algorithms. In *Third International Symposium on Experimental Robotics*, Kyoto.

Moody, J. and Darken, C. J. (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294.

Moore, A. W. (1990). *Efficient Memory-Based Learning for robot control*. PhD thesis, Cambridge University, England. Tech report No. 209.

Morse, T. M., Lockery, S. R., and Ferree, T. C. (1998). Robust spatial navigation in a robot inspired by chemotaxis in caenorhabolitis elegans. *Adaptive Behavior*, 6(3/4):393–410.

Narendra, K. S. and Thathachar, M. A. L. (1989). *Learning Automata: An Introduction*. Prentice-Hall, Englwood Cliffs, NJ.

Nolfi, S., Elman, J. L., and Parisi, D. (1994). Learning and evolution in neural networks. *Adaptive Behavior*, 3(1):5–28.

Oja, E. and Kaski, S., editors (1999). *Kohonen Maps*. Elsevier.

Pavlov, I. (1927). *Conditioned Reflexes*. Oxford University Press, Oxford.

Peng, J. (1993). *Efficient Dynamic Programming-based learning for control*. PhD thesis, Northeastern University, Boston.

Peng, J. and Williams, R. J. (1996). Incremental multi-step q-learning. *Machine Learning*, 22:283–290.

Powell, M. J. D. (1987). *Radial Basis Functions for multivariable interpolation:a review*, pages 143–167. Clarendon Press.

Prescott, A. J. (1994). *Explorations in Reinforcement and Model-based Learning*. PhD thesis, University of Sheffield.

Prescott, T. J., Redgrave, P., and Gurney, K. (1999). Layered control architectures in robots and vertebrates. *Adaptive Behavior*, 7(1):99–127.

Price, B. and Boutilier, C. (2000). Imitation and reinforcement learning in agents with heterogeneous actions. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000)*.

Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning: the rprop algorithm. In *IEEE International conference on neural networks*, pages 586–591.

Ritter, H., Martinetz, T., and Schulten, K. (1990). *Neuronal Netze*. Addison-Wesley.

Ritter, H. and Schulten, K. (1987). Extending kohonen's self-organising mapping algorithm to learn ballistic movements. *Neural Computers*, F41:393–406.

Rojas, R. (1996). *Neural Networks: A systematic introduction*. Springer.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organisation in the brain. *Psychological Review*, 65:386–408.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing*, volume 1. MIT Press, Cambridge, MA.

Rummery, G. A. (1995). *Problem solving with reinforcement learning*. PhD thesis, Cambridge University.

Rummery, G. A. and Niranjan, M. (1994). On-line q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Engineering Department, Cambridge University.

Santamaria, J. C., Sutton, R. S., and Ram, A. (1997). Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior*, 6(2):163–217.

Sehad, S. and Touzet, C. (1994). Self-organising map for reinforcement learning: Obstacle avoidance with khepera. In *Proceedings of From Perception to Action*, Lausanne, Switzerland. IEEE Computer Society Press.

Sejnowski, T. and Rosenberg, C. (1986). Nettalk: A parallel network that learns to read aloud. Technical Report JHU/EECS-86/01, The John Hopkins University, Baltimore, MD.

Sejnowski, T. and Rosenberg, C. (1987). Parallel networks that learn to pronounce english text. *Complex Systems*, 1:145–168.

Shackleton, J. and Gini, M. (1997). Measuring the effectiveness of reinforcement learning for behaviour based robots. *Adaptive Behavior*, 5(4):365–369.

Shiffrin, R. and Schneider, W. (1977). Controlled and automatic human information processing: Ii. *Psychological Review*, 84:127–190.

Shillcock, R. C. and Monaghan, P. (2001). The computational exploration of visual word recognition in a split model. *Neural Computation*, 13:1171–1198.

Singh, S. P. (1992). Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8(3):323–339.

Singh, S. P., Jaakkola, T., Littman, M. L., and Szepesvari, C. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308.

Smith, A. J. (2001). Dynamic actions in reinforcement learning. http://www.dai.ed.ac.uk/homes/andys/PAPERS/papers.html.

SOM-database (2001). http://www.cis.hut.fi/research/som-bibl/.

Sutton, R. S. (1984). *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts.

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

Sutton, R. S. (1990). Integrated architectures for learning, planning and reacting based on approximating dynamic programming. In *Proceedings of seventh international conference on machine learning*, pages 216–224, San Mateo, CA. Morgan Kaufmann.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems: Proceedings of the 1995 conference*, pages 1038–1044, Cambridge, MA. MIT Press.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning*. MIT Press.

Tani, G. J. and Fukumura, N. (1994). Learning goal-directed sensory-based navigation of a mobile robot. *Neural Networks*, 7(3):553–563.

Tani, J., Yamamoto, J., and Nishi, H. (1997). Dynamical interactions between learning, visual attention, and behaviour: An experiment with a vision-based mobile robot. In *Proceedings of the Fourth European Conference on Artificial Life*, pages 309–317.

Tesauro, G. J. (1992). Practical issues in temporal difference learning. *Machine Learning*, 8:257–277.

Tesauro, G. J. (1994). Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219.

Thorndike, E. (1911). *Animal Intelligence*. Hafner, Darien, CT.

Touzet, C. (1997). Neural reinforcement learning for behaviour synthesis. *Robotics and Autonomous Systems. Special Issue on Learning Robots: The New Wave*, 22:251–281.

Tsotsos, J. (1995). Behaviourist intelligence and the scaling problem. *Artificial Intelligence*, 75(2):135–160.

Versino, C. and Gambardella, L. M. (1995). Learning the visiomotor coordination of a mobile robot by using the invertible kohonen map.

Versino, C. and Gambardella, L. M. (1996). Learning fine motion by using the hierarchical extended kohonen map. In von der Malsburg, C., von Seelen, W., Vorbruggen, J. C., and Sendhoff, B., editors, *Artificial Neural Networks—ICANN 96. 1996 International Conference Proceedings*, pages 221–226. Springer-Verlag, Berlin, Germany.

Watkins, C. J. (1989). *Learning from delayed rewards*. PhD thesis, Cambridge university.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3/4):279–292.

Wedel, J. and Polani, D. (1996). Critic-based learning of actions with self-organising feature maps. Technical Report 5/96, Institute for Informatics, Johannes-Gutenberg University.

Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits. *WESCON Convention Record Part IV*, pages 96–104.

Williams, C. (2000). Personal communication. IANC, Edinburgh university.

Williams, R. J. (1988). Towards a theory of reinforcement-learning connectionist systems. Technical Report NU-CCS-88-3, College of Computer Science of Northeastern University, Boston, MA.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256.

Wilson, M. A. and McNaughton, B. L. (1993). Dynamics of the hippocampal ensemble code for space. *Science*, 261:1055–1058.

Ziemke, T. (1996). Towards adaptive behaviour system integration using connectionist infinite state automata. In Maes, P., Mataric, M. J., Meyer, J.-A., Pollack, J., and Wilson, S. W., editors, *From Animals to Animats 4. Proceedings of the fourth international conference on simulated and adaptive behavior*, pages 145–154, Cambridge, Massachesetts. London, England. MIT Press.

Index