Policy-Gradient Algorithms for Partially Observable Markov Decision Processes

Douglas Alexander Aberdeen

A thesis submitted for the degree of
Doctor of Philosophy at
The Australian National University

April 2003

Typeset in Computer Modern by TEX and LATEX 2_ε.

Except where otherwise indicated, this thesis is my own original work.

Douglas Alexander Aberdeen
9 April 2003

Acknowledgements

Academic

Primary thanks go to Jonathan Baxter, my main advisor, who kept up his supervision despite going to work in the "real world." The remainder of my panel was Sylvie Thiébaux, Peter Bartlett, and Bruce Millar, all of whom gave invaluable advice. Thanks also to Bob Edwards for constructing the "Bunyip" Linux cluster and co-authoring the paper that is the basis of Chapter 11.

I thank Professors Markus Hegland, Sebastian Thrun, Shie Mannor, Alex Smola, Brian Anderson and Yutaka Yamamoto for their time and comments. Thanks also to Phillip Williams for mathematical assistance.

Nine of my most productive months were spent at Carnegie Mellon University. Thank-you to the people that made that experience possible, especially Sebastian Thrun and Diane Stidle.

The money came from the Australian government in the form of an Australian Postgraduate Award, and from the Research School of Information Science and Engineering at the ANU.

Sanity

Working on one project for several years is bound to lead to periods when you wish you were doing anything else *but* your project. Thanks to my friends and family for being there to take my mind to happier places during those periods. In particular, my parents Sue and Allan, Hilary McGeachy and "that lot," Belinda Haupt, John Uhlmann, and Louise Sutherland. Also, my two Aussie mates in Pittsburgh who housed me and kept me smiling: Vijay Boyapati and Nigel Tao.

My fellow students must be thanked for their roles as idea sounding boards and sympathetic ears. Especially Cheng Soon Ong, Ed Harrington, and Dave Davies, who all proof-read parts of this thesis.

Abstract

Partially observable Markov decision processes are interesting because of their ability to model most conceivable real-world learning problems, for example, robot navigation, driving a car, speech recognition, stock trading, and playing games. The downside of this generality is that exact algorithms are computationally intractable. Such computational complexity motivates approximate approaches. One such class of algorithms are the so-called policy-gradient methods from reinforcement learning. They seek to adjust the parameters of an agent in the direction that maximises the long-term average of a reward signal. Policy-gradient methods are attractive as a *scalable* approach for controlling partially observable Markov decision processes (POMDPs).

In the most general case POMDP policies require some form of internal state, or memory, in order to act optimally. Policy-gradient methods have shown promise for problems admitting memory-less policies but have been less successful when memory is required. This thesis develops several improved algorithms for learning policies with memory in an infinite-horizon setting. Directly, when the dynamics of the world are known, and via Monte-Carlo methods otherwise. The algorithms simultaneously learn how to act and what to remember.

Monte-Carlo policy-gradient approaches tend to produce gradient estimates with high variance. Two novel methods for reducing variance are introduced. The first uses high-order filters to replace the eligibility trace of the gradient estimator. The second uses a low-variance value-function method to learn a subset of the parameters and a policy-gradient method to learn the remainder.

The algorithms are applied to large domains including a simulated robot navigation scenario, a multi-agent scenario with 21,000 states, and the complex real-world task of large vocabulary continuous speech recognition. To the best of the author's knowledge, no other policy-gradient algorithms have performed well at such tasks.

The high variance of Monte-Carlo methods requires lengthy simulation and hence a super-computer to train agents within a reasonable time. The ANU "Bunyip" Linux cluster was built with such tasks in mind. It was used for several of the experimental results presented here. One chapter of this thesis describes an application written for the Bunyip cluster that won the international Gordon-Bell prize for price/performance in 2001.

Contents

Chapter 1

Introduction

The question of whether a computer can think is no more interesting than the question of whether a submarine can swim.

—Edsger W. Dijkstra

The ultimate goal of machine learning is to be able to place an agent in an unknown setting and say with confidence "go forth and perform well." This thesis takes a step toward such a goal, implementing novel finite-memory policy-gradient methods within the framework of partially observable Markov decision processes.

1.1 Motivation

The framework of partially observable Markov decision processes, or POMDPs, is so general and flexible that devising a "real world" learning problem that *cannot* be cast as a POMDP is a challenge in itself. We now introduce POMDPs by example, exploring the difficulties involved with learning within this framework.

1.1.1 Why Study POMDPs?

To introduce POMDPs let us consider an example based on the work of McCallum [1996], where an agent learns to drive a car in New York. The agent can look forward, backward, left or right. It cannot change speed but it can steer into the lane it is looking at. Observations from the world take multiple forms. One task of the agent is to learn to fuse or ignore different forms as appropriate. Different types of observation in the New York driving scenario include:

- the direction in which the agent's gaze is directed;
- the closest object in the agent's gaze;
- whether the object is looming or receding;
- the colour of the object;
- whether a horn is sounding.

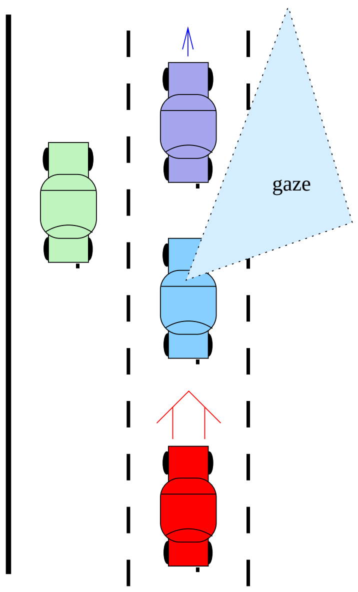


Figure 1.1: The agent is in control of the middle car. The car behind is fast and will not slow down. The car ahead is slower. To avoid a crash the agent must steer right. However, when the agent is gazing to the right, there is no immediate observation that tells it about the impending crash.

To drive safely the agent must steer out of its lane to avoid slow cars ahead and fast cars behind, as depicted in Figure 1.1. This is not easy when the agent has no explicit goals beyond "performing well," and no knowledge of how the observations might aid performance. There are no explicit training patterns such as "if there is a car ahead and left, steer right." However, a scalar reward at each time step is provided to the agent as a performance indicator. The agent is penalised for colliding with other cars or the road shoulder. The only goal hard-wired into the agent is that it must maximise a *long-term* measure of the rewards.

Two significant problems make it difficult to learn under these conditions. The first is solving the *temporal credit assignment* problem. If our agent hits another car and is consequently penalised, how does the agent reason about which sequence of actions should not be repeated, and in which circumstances? For example, it cannot assume that the last action: "change lane," was solely responsible because the agent must change lanes in some circumstances.

The second problem is *partial observability.* If the agent is about to hit the car ahead of it, and there is a car to the left, then circumstances dictate the agent should steer right. However, when the agent looks right in preparation for the lane change, it no longer receives sensory information about the cars ahead or to its left. There is no *immediate* sensory impetus for the lane change. Alternatively, if the agent has *memory* then it builds up knowledge of the *state* of the world around it and it remembers that it needs to make the lane change.

Partial observability takes two forms: absence of important state information in observations and extraneous information in observations. The agent must use memory to compensate for the former, and learn to ignore the latter. For our car driving agent the colour of the object in its gaze is extraneous (unless red cars really do go faster)

If the agent has access to the complete state — such as a chess playing machine that can view the entire board — it can choose optimal actions without memory. This is possible because of the Markov property which guarantees that the future state of the world is simply a function of the current state and action.

This driving example is non-trivial yet easily breaks down into the simple components of a POMDP:

- world state: the position and speed of the agent's car and all other cars;
- observations: hearing a horn honk or observing a slow car ahead;
- memory (agent state): remembering out-of-gaze car locations;
- actions: shifting gaze and steering;
- rewards: penalties for collisions.

To emphasise the generality of the POMDP model, consider the following real-life learning problems in a POMDP framework:

- **Learning to walk**. Babies learn to walk by observing other people do it, imitating them, and being rewarded by the attention of their parents or their ability to reach up to the chocolate cake on the kitchen bench.
- **Learning to understand speech**. Babies learn to understand the sounds they hear and respond in kind. Temporal credit assignment is hard. For example, it is difficult to associate being chastised with the action of stealing the chocolate cake half-an-hour before. These abstract examples can be made more concrete by considering how to teach robots to walk and understand human speech.
- **Maximising the yield of a manufacturing plant** [Smallwood and Sondik, 1973]. Decisions about how often to test manufacturing equipment, or how often to replace parts, affect the yield of the plant. Waiting too long causes expensive delays while failures are fixed. Replacing and testing too often is also costly. Partial observability arises because the presence of faulty equipment might only be

observable through a high failure rate at quality check points. The instantaneous reward is the yield of the plant during the last time period.

The generality of POMDPs makes the learning problem inherently difficult. Markov Decision Processes (MDPs) and early POMDP methods made assumptions that simplified the problem dramatically. For example, MDPs assume the state of the world is known (full observability), or that the dynamics of the world are completely known. As algorithms have become more sophisticated, and computers faster, the number of simplifying assumptions have been reduced.

However, the problem of finding the optimal policy given partial observability is PSPACE-hard [Cassandra, 1998], with exact algorithms running in exponential time and polynomial space in the number of state variables and observations. This motivates the use of three of the approximations applied in this thesis: a fixed number of bits of memory, the use of function approximators for controlling memory and actions, and local optimisation.

Because POMDP algorithms learn from rewards they are sometimes collectively referred to as *reinforcement learning*. Provided the reward signal is constructed carefully, agents will not be biased toward a particular way of achieving the goal; they can learn concepts that their programmers had never thought of. POMDP methods therefore have great potential in domains where human understanding is incomplete or vague. Examples include learning to play games [Tesauro, 1994, Baxter et al., 2001b], network routing [Tao et al., 2001], call admission control [Baxter et al., 2001a], simulating learning in the brain [Barlett and Baxter, 1999], medical diagnosis and treatment [Smallwood et al., 1971], and moving target tracking [Pollock, 1970]. In Chapter 10 we will examine speech recognition as a difficult and novel application of POMDPs.

On the other hand, supervised machine learning methods such as error back propagation only learn concepts embedded in training patterns: they are only as good as their teacher. Tesuro [1990] trained the NeuroGammon system using previously played games as training examples. This system reached a high intermediate level. The use of explicitly supervised learning limited NeuroGammon to tactics captured by the training examples, ensuring it never developed original and superior tactics. The next attempt was TD-Gammon [Tesauro, 1994] which used reinforcement learning. This resulted in a world champion backgammon machine.

1.1.2 Our Approach

We present four algorithms for training agents in the POMDP framework. The first algorithm assumes knowledge of the probability distributions governing world state transitions and observations. The others sit in the hardest class of POMDPs: those where information can only be gathered through interacting with the world and where memory is required to perform well. The algorithms all share three features:

1. **Finite state controllers for agent memory**. The agent has a finite set of memory states, or internal states. When the agent receives an observation it can make a transition from one internal-state to another. During training the agent learns to control these transitions so that the internal states provide useful information about past observations and actions. Key advantages of this method over others are:

   - the ability to remember key events indefinitely;
   - the ability to ignore irrelevant events;
   - per-step complexity that is at worst quadratic in the number of world states, actions, observations, and internal states.

2. **Gradient ascent of the long-term average reward**. The agent's behaviour depends on a set of internal parameters. Our goal is to adjust these parameters to maximise the long-term average reward. We do this by estimating the gradient of the long-term average reward with respect to the agent parameters. Some parameters control the internal-state transitions, and others control the actions the agent chooses at each time step. Key advantages of this method are:

   - convergence to at least a local maximum is assured under mild conditions;
   - the agent learns to choose correct actions without the complex intermediate step of learning the value of each action;
   - system memory usage that is proportional to the number of internal parameters rather than the complexity of the POMDP.

   Potential disadvantages of gradient ascent methods include:

   - many trials or simulations of the scenario may be needed to achieve convergence;
   - we cannot usually guarantee convergence to the global maximum.

   The first disadvantage arises from problems such as high-variance in the gradient estimates. This thesis looks at several methods to reduce the number of trials needed. The second disadvantage is typical of local optimisation and we accept this approximation to reduce the algorithmic complexity.

3. **Infinite-horizon rewards**. The temporal credit assignment problem can be solved in two ways: (1) model the system with at least one recurring state that

bounds the implications of actions on the reward, (2) introduce an "effective horizon" that is tuned to the problem. The first solution corresponds to the finite-horizon assumption, essentially saying that the POMDP is guaranteed to reset itself periodically. The second solution is the infinite-horizon setting, which is more general because it does not require that an *observable* recurrent state can be identified, or even that one exists. The recurrent state must be observable so that the agent can reset its credit assignment mechanism when the state is encountered. Even when observable recurrent states can be identified, they may occur so infrequently that algorithms that use the finite-horizon assumption may perform poorly.

Effective horizons can be introduced by assuming the reward received was in response to a "recent" action. Alternatively, higher order filters can encode information about the credit assignment problem. The most common approach is to assume that rewards are exponentially more likely to be due to recent actions. Effective horizons satisfy our intuition that we can learn to act well by observing a short window of the overall task. Consider learning to play a card game at a casino. The game may go on indefinitely but a few hands are sufficient to teach someone the basics of the game. Many hands give a player sufficient experience to start to reasoning about long-term strategy.

Chapter 2 will review alternative approaches to implementing memory and training agents.

1.2 Thesis Contributions

This section summarises the contributions of the thesis, simultaneously outlining the rest of the document. Chapter 2 provides a formal introduction to partially observable Markov decision processes, describing historical approaches and placing our work in context. Chapter 3 states some mild assumptions required for convergence of stochastic gradient ascent and describes how they can be satisfied. Chapters 4 to 6 introduce our novel policy-gradient algorithms. Chapters 7 and 8 provide some practical implementation details and experimental comparisons. The remaining three chapters are largely independent, presenting variance reduction methods, applications of our policy-gradient methods to speech recognition, and fast training using the "Bunyip" Beowulf cluster. The research contributions of each chapter are:

- **Chapter 4**:
  - We introduce the GAMP algorithm for approximating the gradient of the long-term average reward when a discrete model of the world is available. It

uses a series matrix expansion of the expression for the exact gradient. The algorithm works in the infinite-horizon case, uses finite state controllers for memory, and does not need to gather experience through world interactions. The algorithm takes advantage of sparse POMDP structures to scale to tens-of-thousands of states.

– Using GAMP we demonstrate that a noisy multi-agent POMDP with 21,632 states can be solved in less than 30 minutes on a modern desktop computer.

- **Chapter 5**:
  - This chapter introduces IState-GPOMDP, an algorithm that approximates the gradient when a model of the world is *not* available. In this case learning can only proceed by interacting with the world. It scales to uncountable state and action spaces and operates in an infinite-horizon setting. This is in contrast to similar previous algorithms that are restricted to finite-horizon tasks. The complexity of each step does not depend on the size of the state space and is linear in the number of internal states.
- **Chapter 6**:
  - The IOHMM-GPOMDP algorithm uses hidden Markov models (HMMs) to estimate the state hidden by partial observability. Existing methods that use HMMs ignore the most useful indicator of performance: the reward. IOHMM-GPOMDP learns to predict rewards, thus revealing the hidden state that is relevant to predicting the long-term average reward.
  - IState-GPOMDP gathers experience using a single trajectory through world states and internal states. Exp-GPOMDP is an alternative that still samples world states but takes the expectation over all internal-state trajectories. This reduces the variance of gradient estimates at the cost of making the per step complexity quadratic in the number of internal states.
- **Chapter 7**:
  - The literature devoted to finite state controller (FSC) methods uses toy examples with a few tens of states and one or two bits of memory. We analyse why existing policy-gradient based FSC methods fail to scale to more interesting scenarios. This analysis suggests a method for scaling to larger POMDPs by using *sparse* FSCs.

- **Chapter 8**:
  - We present empirical results on a variety of POMDPs, some larger than previous FSC results in the literature, demonstrating the use of sparse FSC methods on difficult scenarios.
- **Chapter 9**:
  - We introduce two novel methods that reduce the variance of gradient estimates. The first details how to estimate gradients using infinite impulse response filters to encode domain knowledge. The second proposes learning a subset of the parameters with value-function methods to take advantage of their relatively low variance.
  - The application of an existing variance reduction method to our gradient estimates is described. This method uses a fixed sequence of random numbers to perform Monte-Carlo estimates. Our investigation shows that this method must be used with care because it can introduce a form of over-fitting.
- **Chapter 10**:
  - We investigate the use of our algorithms on a large-scale, difficult, real-world problem: speech recognition. We list some advantages of POMDP models over existing speech frameworks. A series of experiments are conducted starting with the simple problem of discriminating binary sequences, moving onto discriminating between spoken digits, and ending with a foray into large vocabulary connected speech recognition (LVCSR). We demonstrate results that are competitive with methods using similar models but trained traditionally.
- **Chapter 11**:
  - The scale of problems such as LVCSR requires the use of super-computing time. Our small budget demanded ingenuity in constructing a cost-effective super-computer. We outline the hardware and software behind the team effort that won a Gordon-Bell prize in 2001, creating the world's first sub USD $1 per MFlop/s super-computer: the "Bunyip" Beowulf-cluster. This cluster was used for many of the experiments in this thesis.

Each chapter contains its own summary and ideas for future work, so we conclude in Chapter 12 with a vision of how disparate state-of-the-art methods for POMDPs could be combined to achieve a powerful and practical learning system.

1.3 Notation

This section describes notation conventions and shortcuts used throughout the thesis. Calligraphic letters refer to a set and an element of that set is represented by the corresponding lowercase letter. A concession to convention is the set of states $\mathcal{S}$ with elements represented by $i \in \mathcal{S}$. When two elements of the same set are referred to, the second element uses the next letter in the alphabet, or a time subscript, to distinguish them. For example, $g \in \mathcal{G}, h \in \mathcal{G}$ or $g_t \in \mathcal{G}, g_{t+1} \in \mathcal{G}$. Summations over multiple sets will often just use the subscripts to indicate which sets the summation is over, for example,

$$\sum_{i,g} \quad \text{means} \quad \sum_{i\in\mathcal{S}}\sum_{g\in\mathcal{G}}.$$

Functions of the form $\mu(u|\theta, h, y)$ should be understood as the probability of event $u \in \mathcal{U}$ as a function of the variables to the right of the '$|$'. If one of the variables is omitted we mean that the probability of u is independent of that variable, for example, $\mu(u|\theta, y)$ means h plays no part in evaluating the probability of u in the context of the surrounding material. The quantities $\theta \in \mathbb{R}^{n_\theta}$ and $\phi \in \mathbb{R}^{n_\phi}$ represent vectors of adjustable parameters.

Much of this work is concerned with state transitions and trajectories of states. The former will usually be discussed with consecutive alphabetic letters, for example, a transition from state i to state j. When talking about trajectories we use the first letter and time subscripts, for example, a transition from state i_t to state i_{t+1}. Thus $\omega(h|\phi, g, y)$ means the same thing as $\omega(g_{t+1}|\phi, g_t, y_t)$ except the former could be any transition in the trajectory and the latter is the transition at time t.

Uppercase letters always refer to matrices. Lowercase letters can refer to scalars or column vectors. Subscripts generally refer to a value at time t, for example, i_t means the value of i at time t. When referring to the value of the i'th element of the vector a at time t we use the notation $a_t(i)$. Occasional exceptions to any of these conventions will be clarified in the text. Page 187 contains a glossary of the main symbols used throughout the text.

Chapter 2

Existing POMDP Methods

Oh yes, and don't forget the other important rule of CS dissertations construction: always include a quote from Lewis Carroll.

—Spencer Rugaber

In this chapter we introduce our view of agents interacting with the world. Past approaches are discussed and we motivate our use of finite-memory policy-gradient methods as a means of training agents. The chapter covers a broad range of algorithms, forming a snapshot of research into POMDPs.

After formalising the framework that we will use throughout this thesis, Section 2.5 discusses exact and approximate methods for solving POMDPs when the underlying POMDP parameters are known. Section 2.6 discusses methods that assume these parameters are *not* known. This thesis describes novel algorithms for both settings. Section 2.7 describes miscellaneous methods for variance reduction and multi-agent settings.

2.1 Modelling the World as a POMDP

Our setting is that of an agent taking actions in a world according to its policy. The agent receives feedback about its performance through a scalar reward r_t received at each time step.

Definition 1. *Formally, a POMDP consists of:*

- $|\mathcal{S}|$ *states* $\mathcal{S} = \{1, \dots, |\mathcal{S}|\}$ *of the world;*
- $|\mathcal{U}|$ *actions (or controls)* $\mathcal{U} = \{1, \dots, |\mathcal{U}|\}$ *available to the policy;*
- $|\mathcal{Y}|$ *observations* $\mathcal{Y} = \{1, \dots, |\mathcal{Y}|\}$*;*
- *a (possibly stochastic) reward* $r(i) \in \mathbb{R}$ *for each state* $i \in \mathcal{S}$*.*

For ease of exposition we assume $\mathcal{S}, \mathcal{Y}$, and $\mathcal{U}$ are finite. Generalisations to uncountably infinite cases are possible for many algorithms, however, the mathematics is considerably more complex and unlikely to deepen our understanding of the underlying

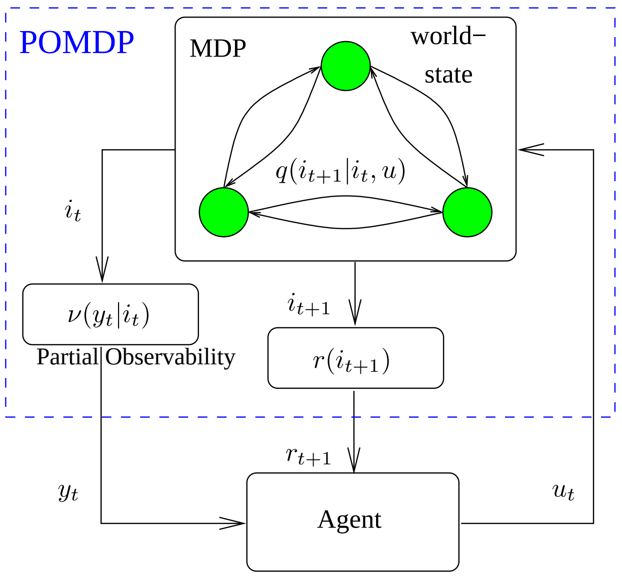


Figure 2.1: Diagram of the world perspective of POMDP showing the underlying MDP, and the stochastic process $\nu(y_t|i_t)$ mapping the current state i_t to an observation y_t, thus hiding the true state information.

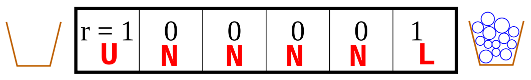
algorithms. Baxter and Bartlett [2001] provide a convergence proof for uncountable state and action spaces for the algorithm that is the parent of those in this thesis.

Each action $u \in \mathcal{U}$ determines a stochastic matrix $[q(j|i,u)]_{i=1\ldots|\mathcal{S}|,j=1\ldots|\mathcal{S}|}$, where $q(j|i,u)$ denotes the probability of making a transition from state $i \in \mathcal{S}$ to state $j \in \mathcal{S}$ given action $u \in \mathcal{U}$. For each state i, an observation $y \in \mathcal{Y}$ is generated independently with probability $\nu(y|i)$. The distributions $q(j|i,u)$ and $\nu(y|i)$, along with a description of the rewards, constitutes the *model* of the POMDP shown in Figure 2.1.

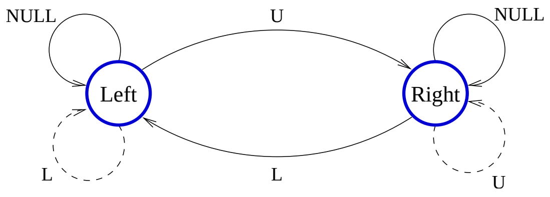
Conceptually, the world issues rewards after each state transition. Rewards may depend on the action and the previous state $r_{t+1} = r(i_t, u_t)$; but without loss of generality we will assume rewards are a function of only the updated state $r_{t+1} = r(j) = r(i_{t+1})$ so that r_1 is the first reward, received after the first action.[1]

If the state is not hidden, that is, there is an observation y for each world state and $\nu(y|i) = 1$ if $y = i$, then we are in the MDP setting. This setting is significantly easier than the POMDP setting with algorithms running in time that is $O(|\mathcal{S}|^2)$ per iteration, or $O(|\mathcal{S}|^3)$ for a closed form solution [Bellman, 1957]. MDP research was the precursor to the study of POMDPs so in Section 2.4 we briefly mention some of the most influential MDP algorithms. POMDPs may be *episodic*, where the task ends upon the agent entering a state i^* which is in the set of termination states $\mathcal{S}^* \subset \mathcal{S}$.

[1]The more general case can be obtained by augmenting the state with information about the last action (and optionally the last state), implicitly making the reward a function of the last action.



(a)



(b)

Figure 2.2: 2.2(a) The Load/Unload scenario with 6 locations. The agent receives a reward of 1 each time it passes through the unload location `U`, or the load location`L`, after having first passed through the opposite end of the road. In each state the agent has a choice of two actions: either **`left`** or **`right`**. The three observations are $\mathcal{Y} = \{$`U, N, L`$\}$, which are issued in the unloading state, the intermediate states and the loading states respectively. 2.2(b) The policy graph learned for the Load/Unload scenario. Each node represents an internal state. The "Left" state is interpreted as: *I have a load so move left,* and the "Right" state as: *I dropped my load so move right.* The dashed transitions are used during learning but not by the final policy.

More generally, they may be infinite-horizon POMDPs that conceptually run forever.

2.2 Agents with Internal State

We have already asserted that agents generally require internal state, or memory, to act well. To make this concept concrete, consider the Load/Unload scenario [Peshkin et al., 1999] shown in Figure 2.2(a). The observations alone do not allow the agent to determine if it should move left or right while it occupies the intermediate `N` observation states. However, if the agent remembers whether it last visited the load or unload location (1 bit of memory) then it can act optimally.

We now introduce a generic internal-state agent model that covers all existing algorithms. We will use it to compare algorithms and introduce our novel algorithms. The agent has access to a set of internal states $g \in \mathcal{G} = \{1, \ldots, |\mathcal{G}|\}$ (I-states for short). Finite memory algorithms have finite $\mathcal{G}$. Alternatively, exact infinite-horizon algorithms usually assume infinite $\mathcal{G}$.[2] For example, one uncountable form of $\mathcal{G}$ is the

[2]It is an abuse of notation to write $g \in \mathcal{G} = \{1, \ldots, |\mathcal{G}|\}$ when $\mathcal{G}$ could be uncountably infinite, however, we ignore this since it is immaterial to the discussion. Our algorithms assume $\mathcal{G}$ is finite.

set of all *belief states*: distributions over world state occupancy in the $|\mathcal{S}|$ dimension simplex. The cross product of the world-state space $\mathcal{S}$ and the internal-state space $\mathcal{G}$ form the *global*-state space.

The agent implements a *parameterised* policy μ that maps observations $y \in \mathcal{Y}$, and I-states $h \in \mathcal{G}$, into probability distributions over the controls $\mathcal{U}$. We denote the probability under μ of control u, given I-state h, observation y, and parameters $\theta \in \mathbb{R}^{n_\theta}$, by $\mu(u|\theta, h, y)$. Deterministic policies emit distributions that assign probability 1 to action u_t and 0 to all other actions. Policies are learnt by searching the space of parameters $\theta \in \mathbb{R}^{n_\theta}$.

The I-state evolves as a function of the current observation $y \in \mathcal{Y}$, and I-state $g \in \mathcal{G}$. Specifically, we assume the existence of a *parameterised* function ω such that the probability of making a transition to I-state $h \in \mathcal{G}$, given current I-state $g \in \mathcal{G}$, observation $y \in \mathcal{Y}$, and parameters $\phi \in \mathbb{R}^{n_\phi}$, is denoted by $\omega(h|\phi, g, y)$. The I-state transition function may be learnt by searching the space of parameters $\phi \in \mathbb{R}^{n_\phi}$.

An important feature of our model is that both the policy and the I-state transitions can be stochastic.

Some algorithms do not learn the I-state transition function, fixing ϕ in advance. Examples include the exact methods of Section 2.5.1, where we can write down the equations that determine the next belief state from the previous belief state and current observation. Algorithms that learn the internal-state transitions include the class of methods we use in this thesis, that is, policy-gradient methods for learning finite state controllers. Throughout this section we compare algorithms in terms of how the $\omega(h|\phi, g, y)$ and $\mu(u|\theta, h, y)$ functions work and how they are parameterised. Tables 2.1 and 2.2 summarise these differences at the end of the chapter.

The agent's view of the POMDP is represented by Figure 2.3. The POMDP evolves as follows:

1. let $i_0 \in \mathcal{S}$ and $g_0 \in \mathcal{G}$ denote the initial state of the world and the initial I-state of the agent respectively;

2. at time step t, generate an observation $y_t \in \mathcal{Y}$ with probability $\nu(y_t|i_t)$;

3. generate a new I-state g_{t+1} with probability $\omega(g_{t+1}|\phi, g_t, y_t)$;

4. generate action u_t with probability $\mu(u_t|\theta, g_{t+1}, y_t)$;

5. generate a new world state i_{t+1} with probability $q(i_{t+1}|i_t, u_t)$;

6. $t = t + 1$, goto 2.

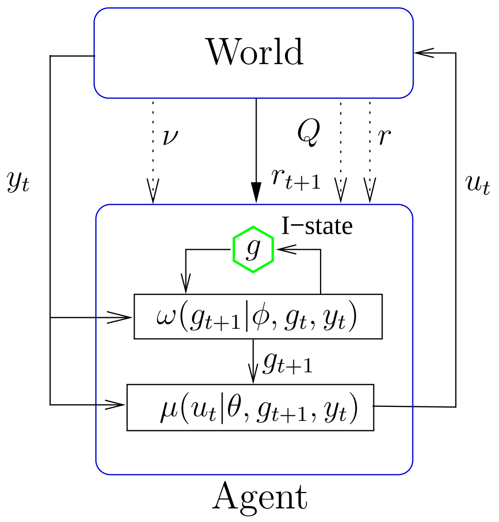

World
ν
Q
r
r_{t+1}
y_t
u_t
I−state
g
$\omega(g_{t+1}|\phi, g_t, y_t)$
g_{t+1}
$\mu(u_t|\theta, g_{t+1}, y_t)$
Agent

Figure 2.3: POMDP from the agent's point of view. The dashed lines represent the extra information available for model-based algorithms.

2.3 Long-Term Rewards

Our algorithms attempt to locally optimise $\theta \in \mathbb{R}^{n_\theta}$ and $\phi \in \mathbb{R}^{n_\phi}$, maximising the *long-term average reward*:

$$\eta(\phi, \theta, i, g) := \lim_{T\to\infty} \frac{1}{T}\mathbb{E}_{\phi,\theta}\left[\sum_{t=0}^{T} r(i_t)|i_0 = i, g_0 = g\right], \tag{2.1}$$

where $\mathbb{E}_{\phi,\theta}$ denotes the expectation over all global trajectories $\{(i_0, g_0), \ldots, (i_T, g_T)\}$ when the agent is parameterised by ϕ and θ. An alternative measure of performance is the *discounted sum of rewards*, introducing a discount factor $\beta \in [0, 1)$

$$J_\beta(\phi, \theta, i, g) := \mathbb{E}_{\phi,\theta}\left[\sum_{t=0}^{\infty} \beta^t r(i_t)|i_0 = i, g_0 = g\right]. \tag{2.2}$$

The exponential decay of the impact of past rewards is equivalent to the assumption that actions have exponentially decaying impact on the current performance of the agent as time goes on. Alternatively, this can be viewed as the assumption that rewards are exponentially more likely to be generated by the most recent actions.

We prefer the long-term average because it gives equal value to all rewards received throughout the evolution of the POMDP. However, the discounted version is useful because it allows the solution of the *temporal credit assignment* problem in infinite-horizon POMDPs by enforcing an effective horizon [Sutton and Barto, 1998]. Fortunately, when the agent is suitably *mixing*, maximising one is equivalent to maximising the other. Suitably mixing means that $\eta(\phi, \theta, i, g)$ is independent of the starting state (i, g) and so may be written $\eta(\phi, \theta)$. In other words, the POMDP is *ergodic*. In this case, there is also a unique stationary distribution over world/internal state pairs. The conditions necessary to ensure ergodicity are mild and will be given in Chapter 3. We denote the expectation over this stationary distribution with $\mathbb{E}_{i,g}$. For ergodic POMDPs the long-term average reward and the discounted reward are related by [Baxter and Bartlett, 2001]

$$\mathbb{E}_{i,g} J_\beta(\theta, \phi, i, g) = \frac{1}{1-\beta} \mathbb{E}_{i,g} \eta(\phi, \theta, i, g).$$

2.4 Methods for Solving MDPs

In this section we describe dynamic programming methods that apply when the state is fully observable, that is, the MDP setting. We will restrict ourselves to the discounted reward setting though the methods can often also be applied to average rewards.

If we have a method of determining the long-term *value* of each state then the agent can act by choosing the action that leads to the state with the highest value. The value is the expected discounted reward for entering state i under the optimal policy. Bellman [1957] describes a procedure known as dynamic programming (DP) which allows us to determine the value $J^*_\beta(i)$ for each state $i \in \mathcal{S}$. We drop the dependency of $J^*_\beta(i)$ on ϕ and θ for this section because DP computes the optimal value directly rather than the value gained by a particular agent. The MDP assumption means that memory is not necessary. This is equivalent to having a single internal state, making the memory process $\omega(h|\phi, g, y)$ trivial. Such agents are said to implement *reactive* policies. DP is described by the *Bellman equation*, where $\beta \in [0, 1)$ is a discount factor that weights the importance of the instantaneous reward against the long-term reward

$$J^*_\beta(i) = \max_u \left[r(i) + \beta \sum_{j \in \mathcal{S}} q(j|i,u) J^*_\beta(j) \right]. \tag{2.3}$$

Assuming we can store a value estimate for each state, DP proceeds by replacing $J^*_\beta(i)$ with an estimate $J_\beta(i)$ and iterating

$$J_\beta(i) \leftarrow \max_u \left[r(i) + \beta \sum_{j \in \mathcal{S}} q(j|i,u) J_\beta(j) \right]. \tag{2.4}$$

In the limit as the number of iterations goes to infinity, $J_\beta(i)$ converges to $J^*_\beta(i)$ [Bertsekas and Tsitsiklis, 1996]. Once $J^*_\beta(i)$ is known the optimal policy is given by

$$u^*_i = \arg\max_u \left[r(i) + \beta \sum_{j \in \mathcal{S}} q(j|i,u) J^*_\beta(j) \right].$$

To reflect full observability, and the absence of internal state, we write the policy as $\mu(u|\theta, i)$ instead of $\mu(u|\theta, h, y)$. For MDPs the optimal μ is deterministic and equal to

$$\mu(u|\theta, i) = \chi_u(u^*_i), \tag{2.5}$$

where $\chi_m(k)$ is the indicator function

$$\chi_m(k) := \begin{cases} 1 & \text{if } k = m, \\ 0 & \text{otherwise.} \end{cases} \tag{2.6}$$

Recall that the vector θ parameterises the policy. In the MDP case θ could represent the value of each state $i \in \mathcal{S}$ or it could represent the mapping of states to actions derived from those values. In the later case θ would be a vector of length $|\mathcal{S}|$, representing a table mapping states directly to the optimal action for that state. When the state and action spaces are large we resort to some form of function approximator to represent the table. For example, the parameters θ could be the weights of an artificial neural network (ANN) that maps states to actions. Function approximation for DP was in use by Bellman et al. [1963] and possibly earlier.

Iterating Equation (2.4) until convergence, and forming a policy from Equation (2.5) is the basis of *value iteration* [Howard, 1960]. Alternatively, *policy iteration* [Bellman, 1957] learns the value of states under a particular agent, denoted $J_{\beta,\mu}(i)$. Once the value of the policy has been learnt the policy is updated to maximise $J_{\beta,\mu}(i)$, followed by a re-evaluation of $J_{\beta,\mu}(i)$ under the new policy. This is repeated until the policy does not change during maximisation. Policy iteration resembles Expectation-Maximisation (EM) methods [Dempster et al., 1977] because we estimate the expected value for state i, then alter the policy to maximise the expected value for i.

Evaluating (2.4) has complexity $O(|\mathcal{U}||\mathcal{S}|^2)$ per step which, while polynomial, is infeasible for very large state spaces. Also, the transition probabilities $q(j|i,u)$, which are part of the model, may not always be available. These two observations motivate Monte-Carlo methods for computing $J^*_\beta(i)$. These methods learn by interacting with the world and gathering experience about the long-term rewards from each state. Q-learning is one algorithm for learning value function $Q(i,u) : \mathcal{S} \times \mathcal{U} \mapsto \mathbb{R}$, which represents the value of taking action u in state i and then acting optimally. It is summarised by the following update rule [Sutton et al., 1999] that introduces a step size

$\gamma_t \in [0, 1)$

$$Q(i_t, u_t) \leftarrow Q(i_t, u_t) + \gamma_t[r_{t+1} + \beta \max_{u'} Q(i_{t+1}, u') - Q(i_t, u_t)],$$

which states that the value $Q(i_t, u_t)$ should be updated in the direction of the error $[r_{t+1} + \beta \max_{u'} Q(i_{t+1}, u')] - Q(i_t, u_t)$. The values converge provided an independent value is stored for each state/action pair, each state is visited infinitely often, and γ is decreased in such a way that $\sum_t \gamma_t = \infty$ and $\sum_t \gamma_t^2 < \infty$ [Mitchell, 1997]. Q-learning is an *off-policy* method, meaning the optimal policy can be learnt by following a fixed alternative policy. This allows a reduction of the number of expensive world interactions by repeatedly re-using experience gathered under an old policy to improve the current policy.

The complexity of learning $J^*_\beta(i)$ or $Q(i, u)$ can be reduced by automatically aggregating states into clusters of states, or meta-states, at the cost of introducing partial observability [Singh et al., 1995, Engel and Mannor, 2001].

Readers interested in MDPs are referred to books such as Puterman [1994], which covers model-based algorithms and many variants on MDPs. Bertsekas and Tsitsiklis [1996] provides an analysis of the convergence properties of MDP algorithms and Kaelbling et al. [1996] describes the algorithms from the *reinforcement learning* point of view.

2.5 Learning with a Model

This section describes existing methods for producing POMDP agents when the model of the POMDP is known, which is equivalent to knowing $\nu(y|i)$, $q(j|i, u)$ and $r(i)$. These include *exact methods* that are guaranteed to learn the optimal policy given sufficient, possibly infinite, time and memory. We define the optimal agent as the agent that obtains the maximum possible long-term (average or discounted) reward given that the agent does not have access to the true state of the system. The long-term reward of optimal MDP methods upper bounds the reward that can be obtained after introducing partial observability [Hauskrecht, 2000].

2.5.1 Exact Methods

The observation process may not reveal sufficient state to allow reactive policies to choose the optimal action. One solution is to model an agent that remembers its entire observation/action history $\bar{y}$. This fits within our framework by setting $\mathcal{G}$ to be the sequence of all past observations and actions $\bar{y} = \{(y_0, u_0), \dots, (y_t, u_t)\}$. Using $\bar{y}$ may allow the agent to determine the true state and hence act optimally. Even if $\bar{y}$ does not allow the agent to determine the true state, it may help to reduce the entropy of the agent's belief of which state it occupies, consequently improving the probability that the agent will act correctly. In this case $\mathcal{G}$ is the possibly infinite set of all observation/action trajectories and $\omega(h|\phi, g, y)$ is a deterministic function that simply concatenates the last observed (y_t, u_t) to $\bar{y}$.

Consider an agent in a symmetric building. If it only receives observations about what is in its line of sight, then many places in the building will appear identical. However, if it remembers the last time it saw a landmark, such as the front doors of the building, then it can infer where it is in the building from its memory of how it moved since seeing the landmark. This is the approach taken by methods such as utile distinction trees [McCallum, 1996] and prediction suffix trees [Ron et al., 1994]. These methods do not necessarily assume knowledge of the POMDP model and will be discussed in Section 2.6.

2.5.1.1 Belief States

Explicitly storing $\bar{y}$ results in inefficient memory use because we potentially need to store an infinite amount of history in infinite-horizon settings. Åström [1965] described an alternative to storing histories which is to track the *belief state*: the probability distribution over world-states, $\mathcal{S}$, given the observation and action history $\bar{y}$. The belief state is a sufficient statistic in the sense that the agent can perform as well as if it had access to $\bar{y}$ [Smallwood and Sondik, 1973]. We use the notation $b_t(i|\bar{y}_t)$ to mean the probability that the world is in state i at time t given the history up to time t. Given a belief state b_t, an action u_t, and an observation y_t, the successor belief state is computed using

$$b_{t+1}(j|\bar{y}_t) = \frac{\nu(y_t|j)\sum_{i\in\mathcal{S}} b_t(i|\bar{y}_t)q(j|i,u_t)}{\sum_{y'\in\mathcal{Y}}\nu(y'|j)\sum_{i\in\mathcal{S}} b_t(i|\bar{y}_t)q(j|i,u_t)}. \tag{2.7}$$

In this setting $\mathcal{G}$ is the possibly uncountable set of belief states the system can reach. From this point on we will use $\mathcal{B}$ instead of $\mathcal{G}$ to refer to the set of reachable belief states, allowing us to keep the belief state distinct from other forms of internal state such as finite state controller I-states. Each element $b \in \mathcal{B}$ is a *vector* in an $|\mathcal{S}|$ dimensional simplex. The function $\omega(b_{t+1}|\phi, b_t, y)$ is deterministic, giving probability 1 to vector b_{t+1} defined by Equation (2.7).

The set $\mathcal{B}$ defines the states of an equivalent MDP on which DP can be performed by replacing the states in Equation 2.3) with belief states [Smallwood and Sondik, 1973, Cassandra et al., 1994]

$$\bar{J}_\beta(b') \leftarrow \max_u \left[\bar{r}(b') + \beta \sum_{b \in \mathcal{B}} \bar{q}(b|b', u) \bar{J}_\beta(b) \right], \tag{2.8}$$

which converges in finite time to within ϵ of the optimal policy value [Lovejoy, 1991]. The bar on $\bar{J}_\beta(b)$ indicates that we are learning values of belief states b instead of world states i. The simplicity of this equation is misleading because $\bar{J}_\beta$, $\bar{r}$, and $\bar{q}$ are represented in terms of their MDP counterparts in Equation (2.3). For example, $\bar{r}(b) = \sum_{i \in \mathcal{S}} b_i r(i)$. When the set of reachable belief states can be infinite, $\bar{J}_\beta(b)$ is significantly more complex than $J_\beta(i)$, which is the subject of the next section.

2.5.1.2 Value Function Representation

Maintaining and updating independent values for infinitely many belief states is infeasible. Fortunately, it has been shown for finite horizons that the value over all belief states can be represented exactly by a convex piecewise linear function [Smallwood and Sondik, 1973]. Such a representation is shown for a 2 state system in Figure 2.4. The set $\mathcal{L}$ contains the hyperplanes needed to represent the value function. Each $l \in \mathcal{L}$ is an $|\mathcal{S}|$ dimensional vector such that the value of hyperplane l for belief state b is $b \cdot l$. The value of $\bar{J}_\beta(b)$ is the maximum over all hyperplanes

$$\bar{J}_\beta(b) = \max_{l \in \mathcal{L}} \{b \cdot l\}.$$

To be useful, a hyperplane l must be the maximum for some b. If $\mathcal{L}$ contains only useful hyperplanes then it is called a *parsimonious* set [Zhang, 1995].

Estimating the value function proceeds by using value iteration on belief states chosen to be points in the belief simplex that have a unique maximuml l, and that have not yet converged to their true value. These points, as found by the Witness algorithm, are called *witness points* [Cassandra et al., 1994]. This term is now often used to describe any belief state upon which it is useful to perform a DP update. Such updates generate new vectors to be added to $\mathcal{L}$. A difficult task — and the way in which most exact POMDP algorithms differ — is determining the witness points efficiently. This is often done by solving a set of linear programs. Once a single round of value iteration has been performed $\mathcal{L}$ is examined to remove useless vectors, creating a parsimonious set ready for the next round.

The simplest exact method is Sondik/Monahan's enumeration algorithm [Monahan, 1982], which enumerates all hyperplanes in $\mathcal{L}$ and performs a DP update on each one. This method results in a large number of new vectors. In rough order of increasing

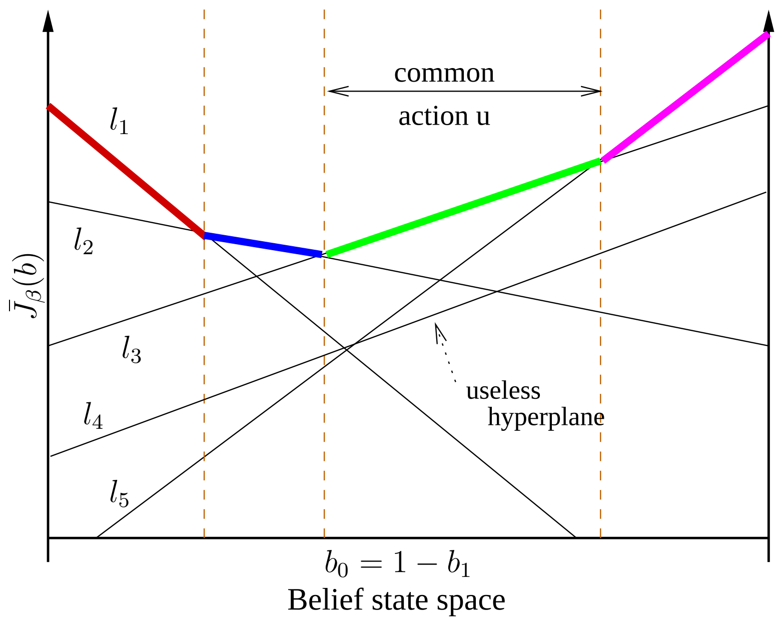


Figure 2.4: Convex piecewise linear representation of a value function for a continuous belief state with $|\mathcal{S}| = 2$. The plot is a slice from a 3-dimensional plot defined by valid points in the the 2-dimensional simplex, that is, where $b_0 = 1 - b_1$. If the set $\mathcal{L}$ is parsimonious then each l_n defines a line segment which is maximum in some partition of belief space. Each such partition has a unique optimal action u_n^*.

sophistication other algorithms include the One-Pass algorithm of Sondik [1971], the Linear Support algorithm of Cheng [1988], the Witness algorithm of Cassandra et al. [1994], and the Incremental Pruning algorithm of Zhang and Liu [1996]. Cassandra [1999] provides a non-mathematical comparison of these algorithms.

2.5.1.3 Policy Representation

POMDP agents can be represented by a policy graph that is a directed and possibly cyclic graph. Each node is labelled with a single action and transitions out of each node are labelled with observations. All nodes map to a polyhedral partition in the value function. Partitions are defined by the region where a single hyperplane $l \in \mathcal{L}$ maximises $\bar{J}_\beta(b)$ [Cassandra et al., 1994]. Transitions between nodes in the policy graph are a deterministic function of the current observation. Actions are a deterministic function of the current node. If the optimal policy can be represented by a cyclic graph with a finite number of nodes, then the POMDP is called *finitely transient* [Sondik, 1978].

A policy can be represented by $\mu(u|\theta, h, y)$ in several ways. The parameters θ could be used to store hyperplanes in $\mathcal{L}$, in which case $\mu(u|\theta, b_{t+1}, y_t)$ gives probability 1 to the action associated with the hyperplane that maximises the value at b_t. If the policy is finitely transient it can be represented more compactly by the policy graph. In this case the internal states $\mathcal{G}$ are equivalent to the policy-graph nodes. The I-state (internal state) update $\omega(h|\phi, g, y)$ uses y to index the correct transition from node g to node h. The policy $\mu(u|\theta, h)$ is simply a lookup table that gives the optimal action for each node h. Thus, ϕ represents policy-graph transitions, and θ maps policy-graph nodes to actions.

In many cases it is possible for large infinite-horizon POMDPs to be controlled well by simple policy graphs. Consider the Load/Unload scenario [Peshkin et al., 1999] shown in Figure 2.2(a). The optimal policy graph is shown in Figure 2.2(b). This policy graph suffices no matter how many intermediate locations there are between the load and unload locations. As the number of intermediate locations increases, the value function becomes more complex but the optimal policy graph does not. This example partially motivates the idea (discussed in Section 2.5.7) of searching in the space of policy graphs instead of learning value functions.

2.5.1.4 Complexity of Exact Methods

There are two problems with exact methods that make them intractable for problems with more than a few tens of states, observations, and actions. To discuss the first we introduce the concept of *state-variables*. State variables describe the state in terms of features that are true or false, which is an arguably more intuitive description than enumerating states. Consider a system with v boolean state variables. The number of POMDP states is $|\mathcal{S}| = 2^v$, thus $|\mathcal{S}|$ grows exponentially with the number of state variables. For example, two state variables might be "is it raining?" and "is the umbrella open?" requiring 4 states to encode.

Since DP for POMDPs involves updating belief states, the complexity of POMDP algorithms grows exponentially with the number of state variables. This makes belief-state monitoring infeasible for large problems [Boyen and Koller, 1998, Sallans, 2000].

The second problem is representing $\bar{J}_t(b)$, the value function after t steps of DP. Let $\mathcal{B}_t$ be the set of belief states reachable at time t. Recall that $|\mathcal{Y}|$ is the number of possible observations. Assuming $|\mathcal{B}_0| = 1$, that is, a single known initial belief state, then after t steps of a greedy policy we potentially have $|\mathcal{Y}|^t$ belief states in $\mathcal{B}_t$. Thus, the problem of representing $\bar{J}_\beta(b)$ grows exponentially in the horizon length. Since the belief-state space is infinite for infinite horizons, exact methods perform DP on the hyperplanes in $\mathcal{L}$. This representation grows exponentially in the observations since a single DP step in the worst case results in $|\mathcal{L}_{t+1}| = |\mathcal{U}||\mathcal{L}_t|^{|\mathcal{Y}|}$ [Cassandra, 1998]. Furthermore, evaluating if there exists a belief state b for which an $l \in \mathcal{L}_{t+1}$ is dominant requires solving expensive linear programs.

Even for simplified *finite*-horizon POMDPs, the problem of finding the optimal policy is PSPACE-hard [Papadimitriou and Tsitsiklis, 1987]. Learning a policy graph with a *constrained* number of nodes is NP-hard [Meuleau et al., 1999a]. Infinite-horizon POMDPs can result in an infinite number of belief states or hyperplanes l, resulting in the problem of determining convergence being undecidable [Madani et al., 1999]. Even worse, determining ϵ-convergence is undecidable in polynomial time for *finite*-horizon POMDPs [Lusena et al., 2001]. Despite this gloomy theoretical message, empirical results, such as those in this thesis, show that it is possible to learn reasonable policies in reasonable time for POMDPs of ever increasing complexity. We avoid the intractable computational complexity of exact methods by abandoning the requirement that policies be optimal, while retaining the requirement that agents should at least converge to a fixed policy.

2.5.2 Approximate Value Function Methods

This section introduces model-based methods that learn approximations to $J^*_\beta(i)$. For a more detailed survey of these methods see Hauskrecht [2000].

2.5.2.1 Heuristics for Exact Methods

A number of methods simplify the representation of $\bar{J}_\beta(b)$ by assuming the system is an MDP and learning the underlying Q-function $Q(i, u)$. This must be done via model-based methods or by computer simulation because the partial observability of the real world does not allow i to be known during real-world interaction.

One choice for the policy is [Nourbakhsh et al., 1995]

$$\bar{u}^*(b) = \arg\max_u Q(\arg\max_j b(j), u), \qquad \mu(u|\theta, b) = \chi_u(\bar{u}^*(b)),$$

which assumes the agent is in the most likely state (MLS), known as the MLS heuristic. This approach completely ignores the agent's confusion about which state it is in. The voting heuristic [Simmons and Koenig, 1995] weights the vote for the best action in each state by the probability of being in that state

$$u^*(j) = \arg\max_a Q(j, a), \qquad \bar{u}^*(b) = \arg\max_u \sum_{j \in \mathcal{S}} b(j)\chi_u(u^*(j))Q(j, u^*(j)).$$

The popular Q_{MDP} heuristic [Littman et al., 1995] takes into account the belief state for one step and then assumes that the state is entirely known [Cassandra, 1998]

$$\bar{u}^*(b) = \arg\max_u \sum_{j \in \mathcal{S}} b(j) Q(j, u).$$

These heuristics will perform poorly if the belief state is close to uniform. Due to the convexity of the value function, $\bar{J}_\beta(b)$ generally grows as the entropy of b decreases. The highest expected payoffs occur at the simplex corners. This motivates choosing actions that decrease the entropy of the belief state in the hope that the heuristics above will perform better with a peaked belief. For example, consider a robot that must reach the other side of a featureless desert [Roy and Thrun, 2001]. If it goes straight across it will quickly become lost due to lack of landmarks and movement errors. The better policy is to skirt the desert, taking longer but remaining certain of reaching the goal because the robot is certain of its location. Cassandra [1998] shows how to use the entropy of b to switch between *information gathering* policies and exploitive policies. The entropy can also be used to weight two policies that trade off information gathering and exploitation. These heuristics may be misleading if the minimum of $\bar{J}_\beta(b)$ does not occur at the point of greatest entropy, that is, the uniform belief state.

An alternative family of heuristics are based on simplified versions of the full DP update. For example, determining maximising vectors in $\mathcal{L}$ for each state, rather than over all states, produces the *fast informed bound* of Hauskrecht [1997]. The heuristics in this section are also useful as upper bounds on J^*_β, allowing them to direct tree search procedures used in classical planning approaches (see Section 2.5.4).

2.5.2.2 Grid Methods

Value functions over a continuous belief space can be approximated by values at a finite set of points along with an interpolation rule. Once a set of grid points has been chosen an equivalent MDP can be constructed where the states are the grid points. This POMDP can be solved in polynomial time [Hauskrecht, 2000]. The idea is equivalent to constructing a policy graph where each node is chosen heuristically to represent what might be an “interesting” region of belief state space. The two significant issues are how to choose grid points and how to perform interpolation.

Regular grids are an obvious choice but they fail to scale for large state spaces [Lovejoy, 1991]. Methods for choosing irregular grids include the use of simulation to determine useful grid points [Hauskrecht, 1997] and adding points where large variations in values are detected for two local points that have observations in common

[Brafman, 1997].[3]

Interpolation schemes should maintain the convex nature of the value function and hence are typically of the form

$$\bar{J}_\beta(b) = \arg\max_u \sum_{g\in\mathcal{G}} \lambda_{g,b} f(g,u), \tag{2.9}$$

where $\sum_{g\in\mathcal{G}} \lambda_{g,b} = 1$ and $\lambda_{g,b} \geq 0\ \forall g, b$. The function $f(g,u)$ represents the value grid point g under action u. Examples include nearest neighbour, linear interpolation, and kernel regression [Hauskrecht, 2000]. The MLS heuristic can be thought of as a grid method with points at the belief-simplex corners and a simple 1-nearest-neighbour interpolation (assuming an L_∞ distance measure) [Brafman, 1997].

Recently, ϵ-convergence was shown for a grid method by using ϵ-covers under a specifically chosen distance metric [Bonet, 2002]. Within our knowledge it is the only provably optimal grid based algorithm. It is still intractable because the number of grid points required grows exponentially with the size of the state space.

Alternatively, grid point value estimation steps can be interleaved with exact hyperplane DP updates in order to speed up DP convergence without sacrificing policy optimality [Zhang and Zhang, 2001].

In the context of grid methods the $\omega(b_{t+1}|\phi, b_t, y_t)$ process still performs a deterministic update on the belief state, but now $\mu(u|\theta, b_{t+1}, y_t)$ represents the choice of action based on the interpolated value from (2.9). The θ parameters store the values of actions at the grid points.

2.5.3 Factored Belief States

Using the state variable description of a POMDP (discussed in Section 2.5.1.4) is sometimes referred to as belief factorisation, especially when the factorisation is not exact. The value of a state variable X_t may depend (approximately) on a small subset of the state variables at the previous time steps. Transition probabilities can be represented by a two-slice temporal Bayes net (BN) that models the dependencies between state variables over successive time steps as a 2 layer acyclic graph [Dean and Kanazawa, 1989] (see Figure 2.5). Each node contains a conditional probability table showing how its parents affect the probability of the state variable being true.

Alternatively, the state variable dependencies can be represented by a tree structure such as an *algebraic decision diagrams* (ADD) [Bahar et al., 1993]. BNs and ADDs are applied to POMDPs by Boutilier and Poole [1996] to simplify both the belief monitoring

[3]The mutual observations requirement is an interesting heuristic that may identify whether belief states between the two existing points can be reached.

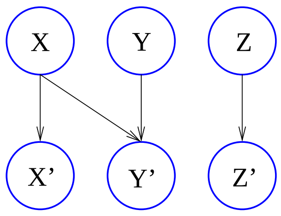


Figure 2.5: Two-slice temporal Bayes network showing dependencies between state variables over successive time steps. X and Z depend only on themselves but Y is influenced by itself and X. This conditional dependence structure can be exploited to efficiently model the transition probabilities of a POMDP.

problem and the value function representation problem. We now describe how recent work has used belief factorisation to solve each of these problems.

2.5.3.1 Efficiently Monitoring Belief States

Boyen and Koller [1998] observed that representing belief states as BNs can lead to an accumulation of errors over many time steps that result in the belief state approximation diverging. The authors show that projections of the BN that produce strictly independent groups of state variables results in converging belief-states, allowing recovery from errors. These projections can be searched to automatically determine BNs that perform well under a specified reward criterion. Heuristic methods to determine factorisations were introduced by Poupart and Boutilier [2000]. These were improved in Poupart and Boutilier [2001] which presents a vector-space analysis of belief-state approximation, providing a formally motivated search procedure for determining belief-state projections with bounded errors. This expensive off-line calculation speeds up on-line belief state tracking.

An alternative method of searching for the optimal belief factorisation is to learn a dynamic sigmoid belief network [Sallans, 2000]. Stochastic gradient ascent is performed in the space of BNs with a fixed number of dependencies, minimising the error between the belief state and the approximation. This algorithm has been applied to the large New York driving problem of McCallum [1996] which was sketched out in Section 1.1. Unlike the algorithms considered so far, which apply a fixed I-state update through $\omega(h|\phi, g, y)$, this algorithm can be viewed as learning $\omega(h|\phi, g, y)$ by adjusting the belief network parameterised by ϕ.

2.5.3.2 Factored Value Functions

So far we have discussed piecewise linear value functions represented by sets of hyperplanes $\mathcal{L}$. Koller and Parr [2000] note that even if compact BN representations of the transition model are found, they may not induce a similar structure on the value function. The same paper discusses factored value functions implemented as weighted linear combinations of polynomial basis functions. The weights are chosen to minimise the squared error from the true value. In our model of agents the weights would be represented by θ.

This idea is combined with BNs for belief monitoring by Guestrin et al. [2001a]. With approximate belief monitoring and factored linear value functions, DP methods become more feasible since learning reduces to solving systems of linear equations instead of a linear program for every vector in $\mathcal{L}$. The empirical advantages of factored value functions are studied in [Hansen and Feng, 2000] which uses algebraic decision diagrams to represent $\bar{J}_\beta(b)$.

2.5.4 Classical Planning

The link between classical planning algorithms such as C-Buridan [Draper et al., 1994] and POMDP algorithms is summarised by Blythe [1999]. One planning approach is to search the tree of possible future histories $\bar{y}$ to find the root action with the highest expected payoff. Branch and bound approaches such as the AO^* algorithm [Nilsson, 1980, §3.2] can be used, searching the tree to a finite depth and pruning the search by eliminating branches that have upper bounds below the best current lower bound. At the leaves the upper and lower bounds can be estimated using Q_{MDP} and the value-*minimising*-action respectively [Washington, 1997].

This idea is applied to *factored* finite-horizon POMDPs by McAllester and Singh [1999]. While avoiding the piecewise linear value function, the algorithm is still exponential in the horizon due to the tree search procedure. Furthermore, since there is no concise policy representation the exponential complexity step is performed during normal policy execution instead of only during a learning phase. In this case $\omega(b_{t+1}|\phi, b_t, y_t)$ is responsible for deterministically tracking the factored belief state, and $\mu(u|\theta, b)$ is responsible for performing the tree search in order to find the best u. The parameters θ are not learnt, but ϕ could be learnt to optimise factored belief state tracking, as in Section 2.5.3.1.

2.5.5 Simulation and Belief States

The full DP update of Equation (2.8) is inefficient because all states are assumed equally important. Simulation algorithms learn value functions from real experience, concentrating learning effort on the states most likely to be encountered. This idea can be applied to learning Q-functions for finite-horizon POMDPs by simulating a path through the POMDP while monitoring the current belief state and performing an iteration of (2.8) for the current point in belief state space instead of the associated $l \in \mathcal{L}$ [Geffner and Bonet, 1998]. This algorithm, called RTDP-Bel, does not address the problem of representing $Q(b, u)$ for many belief states over large horizons. However, it has been applied to solving finite-horizon POMDPs with hundreds of states and long-term memory. In particular, it has been applied to the Heaven/Hell scenario that we shall investigate in Chapter 8.

Usually we need to learn Q-functions that generalise to all b. A simple approach is Linear-Q which defines $Q(b, u) = \phi_u \cdot b$ [Littman et al., 1995]. This method is equivalent to training a linear controller using stochastic gradient ascent with training patterns given by inputs b_t and target outputs $r + \beta \max_u Q(b_{t+1}, u)$. The SPOVA algorithm [Parr and Russell, 1995] is essentially the same scheme, using simulation to generate gradients for updating the smooth approximate function

$$\bar{J}_\beta(b) = \left[\sum_{l \in \mathcal{L}} (b \cdot l)^k\right]^{\frac{1}{k}},$$

where k is a tunable smoothing parameter and the number of hyperplanes $|\mathcal{L}|$ is fixed. Artificial neural networks (ANNs) can also be used to approximate value functions with either the full belief state or a factored representation used as the input [Rodríguez et al., 2000]. Both Linear-Q and SPOVA learn the θ parameters but not ϕ.

2.5.6 Continuous State and Action Spaces

Thrun [2000] extends value iteration for belief states into continuous state spaces and action spaces. Since infinite memory would be needed to exactly represent belief states, Thrun uses a sampled representation of belief states. The algorithm balances accuracy with running time by altering the number of samples n used to track the belief state. The belief state is updated using a particle filter [Fearnhead, 1998] that converges to the true belief state as $n \to \infty$, for arbitrary continuous distributions $q(j|i, u)$ and $\nu(y|i)$. Approximate continuous belief states are formed from the samples using Gaussian kernels. The hyperplane value-function representation cannot be used because the belief state is a continuous distribution, equivalent to infinitely many belief state dimensions. Even function approximators like neural networks cannot trivially contend with continuous belief states. Instead, the value function is approximated using a nearest neighbour method where the output value is the average of the k nearest neighbours (KNN) of previously evaluated infinite-dimension belief states.

Particle filters can also be used to monitor the belief state during policy execution. Poupart et al. [2001] analyses the value loss incurred by using particle filters once a value function has already been learnt. The paper uses this analysis to develop an adaptive scheme for choosing n based on the probability of choosing an approximately optimal action. These methods may also help in finite domains with many states.

2.5.7 Policy Search

The approaches we have discussed so far have learnt values for each state and action. Values can be explicitly encoded in θ or used to manufacture a more compact policy graph. This section introduces methods that learn policies directly, sometimes avoiding value estimation at all. For example, we can attempt to learn a set of parameters θ that directly encodes a policy graph.

There are several reasons to prefer policy search methods to value function methods, particularly in the presence of partial observability. Informally, Occam's Razor can be invoked in favour of policy search methods by noting that it is intuitively simpler to determine *how to act* instead of the *value of acting.* For example, Section 2.5.1.3 demonstrated that a 2 node policy graph can optimally control an arbitrarily large POMDP. The value function representation would require storing or approximating a value for each state.

More formally, approximate value function methods can perform poorly in POMDP settings because they usually find deterministic policies (by application of the max function in Equation (2.8)). Approximate POMDP methods generally require stochastic policies [Singh et al., 1994]. Value methods in function approximation settings also lack convergence guarantees because small changes to state values can cause discontinuous changes to the policy [Bertsekas and Tsitsiklis, 1996].

However, policy search is still difficult. Even for restricted classes of policies, the search is NP-hard [Meuleau et al., 1999a]. Value function methods have the advantage of imposing the useful Bellman constraint. Consequently, value functions are likely to remain useful for small to medium size problems.

Policy search can be implemented using *policy iteration* [Bertsekas and Tsitsiklis, 1996, §2.2.3]. The first step is to evaluate the current policy and the second step is policy improvement. Sondik [1978] described how policy evaluation could be performed by representing the policy as a set of polyhedral partitions in belief space. Instead, Hansen [1998] uses the policy-graph representation, greatly simplifying the task of evaluating the policy. Evaluating the policy is performed by solving a set of linear equations that gives the hyperplane representation of the value function $\bar{J}_\beta(b)$. Policy improvement works by adding or merging policy graph nodes based on how the set of vectors in $\mathcal{L}$ changes after a single hyperplane update on the evaluated policy. Hansen also describes a branch and bound tree search procedure that can be used to approximate the DP step, achieving an order of magnitude speed-up.

The nodes of a policy graph can be interpreted as states of an internal MDP that is observable by the agent. The process defined by the cross product of world states with policy-graph states is Markov [Hansen, 1998]. The value of the cross product MDP can be computed using value iteration. Meuleau et al. [1999a] finds optimal policy graphs within a constrained space using a branch and bound search method, computing the value of complete policies at the leaves only when needed.

The same paper also shows how gradient ascent can be used to search the space of stochastic policy graphs, which we refer to as *finite state controllers* (FSCs). FSCs are described in detail in Section 2.6.2.5. Given a model, the gradient of the *discounted* sum of rewards can be computed directly, requiring large matrix inversions. This is the first algorithm discussed so far that does not evaluate values explicitly. Because it does this with gradient ascent it is our first example of a *policy-gradient* method. It is also our first example of a stochastic internal-state function.

The simulation methods of Section 2.5.5 can be used to generate policies instead of value functions. For example, if the belief state is tracked during simulation and used as an input to a neural network, then network outputs can be mapped to a distribution over actions. We demonstrate this method in Chapter 8. We will use policy search methods throughout this thesis.

2.5.8 Hybrid Value-Policy Methods

MDP Value methods may fail to converge for POMDPs or when using function approximation in the MDP setting. In contrast, policy-gradient methods converge to a local maximum under mild conditions. However, policy-gradient methods exhibit high variance [Marbach and Tsitsiklis, 2000]. One reason for high variance is that they ignore the Bellman equation which is a useful constraint for agents [Peshkin et al., 1999, Baird and Moore, 1999]. In an attempt to combine the low variance of value based methods with the convergence to local maximum guarantees of policy-gradient methods, Baird and Moore [1999] introduced VAPS. This algorithm specifies a parameter $\rho \in [0, 1]$ that gives a pure Q-learning algorithm when $\rho = 0$ — satisfying the Bellman equation — and a pure policy search algorithm when $\rho = 1$ — locally maximising the long-term reward using the Williams' REINFORCE algorithm [Williams, 1992] described in Section 2.6.3. Intermediate values of ρ invoke an hybrid algorithm that is guaranteed to converge even for POMDPs.

However, it is not clear how to choose ρ given a POMDP, and because VAPS is not a pure gradient ascent it may not converge to a *locally optimal* policy when $\rho < 1$ [Sutton et al., 2000]. An alternative introduced by Sutton et al. [2000], which does converge to a locally optimal policy, incorporates a learnt Q-function directly into the gradient estimation of $\nabla \bar{J}_\beta$ or $\nabla \eta$. In this case the Q-function works like a critic aiding the actor to learn a policy [Konda and Tsitsiklis, 2000].

Both Williams' REINFORCE and VAPS were developed in the context of solving MDPs using function approximation. They can be extended to POMDPs either by assuming that observations map directly to world states — implying the assumption that a reactive policy is sufficient — and learning the policy $\mu(u|\theta, y)$, or by using the model and $\omega(b_{t+1}|\phi, b_t, y_t)$ to track belief states that are fed to the hybrid policy represented by $\mu(u_t|\theta, b_{t+1})$.

Coarse grained value estimates can be used as a seed for fine grained policy-search methods, avoiding the disadvantages of the curse of dimensionality for DP and the high variance of policy-search in the early stages. This approach is demonstrated on a real robot using a continuous time model by Roy and Thrun [2001].

Konda and Tsitsiklis [2000] present hybrid algorithms as actor-critic algorithms. The actor uses a policy-gradient approach to learn a policy as a function of a low-dimension projection of the exact value function. The projection of the value function is learnt by the critic. Convergence is shown for arbitrary state/action spaces with linear critics. The paper shows that the value function projection that should be used by the critic is determined by the how the actor is parameterised.

2.6 Learning Without a Model

From this point on we do not assume knowledge of the underlying MDP dynamics $q(j|i, u)$, the observation probabilities $\nu(y|i)$, or the rewards $r(i)$. To learn good policies the agent interacts with the world, comparing observation/action trajectories under different policies. With enough samples, rewards can be correlated with actions and the probability of choosing good actions increased. Section 2.4 briefly described Q-learning which is one example of a model-free algorithm for MDPs. It is also a starting point for model-free POMDPs.

2.6.1 Ignoring Hidden State

We can modify MDP Q-learning to learn the value of observations instead of states, that is, $Q(y, u)$ instead of $Q(i, u)$. This is equivalent to learning values over distributions of states instead of single states [Singh et al., 1994]. Such reactive policies generally need to be stochastic. If the agent is uncertain about the state then it may be optimal to choose actions stochastically. Examples can be constructed for which a stochastic reactive policy is arbitrarily better than the optimal deterministic reactive policy [Williams and Singh, 1999]. This method is used with stochastic-policy iteration by Jaakkola et al. [1995], converging to a local maximum when $Q(y, u)$ is parameterised by a table. A similar result is shown [Singh et al., 1994] that also has convergence results for a modified TD(0) MDP algorithm [Sutton and Barto, 1998].

In this memory-less setting $|\mathcal{G}| = 1$ making $\omega(h|\phi, g, y)$ trivial. Consequently, the policy reduces to $\mu(u|\theta, y)$.

2.6.2 Incorporating Memory

When the POMDP model is not available agents cannot track the belief state. The alternative to reactive policies is to use memory of the past observations and actions to resolve the hidden world state. This section describes several mechanisms for adding memory to agents.

2.6.2.1 HMM Based Methods

Solving POMDPs can be re-cast as the problem of determining the hidden state of the world and then applying MDP methods. A natural approach is to use hidden Markov models which have been successfully applied to many other hidden state estimation problems [Poritz, 1988]. Hidden Markov models are briefly introduced in Appendix D. This approach was taken by Chrisman [1992] where HMMs are used to predict observations based on the observation/action history $\bar{y}$. A nice feature of this algorithm is the ability to grow the number of states in the HMM until performance stops improving. HMMs for predicting observations were also studied by McCallum [1996].

A problem with this approach is that the hidden state revealed by modelling *observations* is not guaranteed to reveal the state necessary to maximise the reward. One alternative studied in this thesis uses HMMs to model *rewards*. Also, gradient ascent of HMMs can be used to model *action* generation [Shelton, 2001a,b]. All of the HMM methods discussed in this section use a generalisation of HMMs introduced by Bengio and Frasconi [1995] called *Input/Output* HMMs (IOHMMs). The state transitions of IOHMMs can be driven by a signal other than the one being modeled.

In these models $\mathcal{B}$ is the set of reachable belief states over HMM states. This *internal-state belief* is equivalent to the forward probability over states α, used during Baum-Welch training of HMMs [Rabiner and Juang, 1986]. The probability of occupying each HMM state is updated using $\omega(h|\phi, g, y)$ given current assumed internal belief g and input observation y (or the previous action if observations are being modeled).

The policy $\mu(u|\theta, h, y)$ maps the belief state over HMM states to actions. For example, if the HMM is trained to emit actions then $\mu(u|\theta, h, y)$ represents the emission probabilities of HMM-state h [Shelton, 2001a]. In this case ϕ parameterises HMM transition probabilities and θ parameterises emission densities for predicted observations or generated actions. The internal-state belief update is deterministic, but the policy is stochastic. Alternatively, $\mu(u|\theta, h, y)$ can be learnt by DP, estimating values of internal-state beliefs. This is the approach used by Chrisman [1992] where the Q_{MDP} heuristic is used to assign values to internal-state beliefs.

2.6.2.2 Finite History Methods

Finite history methods produce policies that are functions of a finite window of past observations/actions. They have existed in the context of POMDPs at least since Brown [1972]. Window-Q uses an artificial neural network (ANN) to learn Q-values where the ANN inputs are the last n observations and actions [Lin and Mitchell, 1992]. Time-delay neural networks incorporate n steps of history into each node of the neural network [Lin, 1994]. In either case $\omega(h|\phi, g, y)$ deterministically records a finite window of $\bar{y}$, and $\mu(u|\theta, \bar{y})$ uses an ANN parameterised by θ to assign probability 1 to the action that maximises the value.

Adaptive history methods grow or shrink the number of past events that are needed to reveal the hidden state. Probabilistic Suffix Automata [Ron et al., 1994] model partially observable Markov decision processes using a tree. The history defines a path through the tree starting at the root. The root corresponds to the earliest history element. The leaf can be used to predict the next symbol, or it can be labelled with the optimal next action given the history. McCallum [1996] uses the latter approach in the UTREE algorithm (see Figure 2.6) to control large POMDPs such as the New York Driving problem. A statistical test is applied to leaf nodes to determine if the long-term reward can be improved by growing the tree, automatically determining the amount of memory needed. Observation branches can be selected by a subset of the observation vector, filtering out useless distinctions between observations. Tree methods have not been extended to continuous observation/action spaces within our knowledge. Here $\mu(u|\theta, \bar{y})$ applies the UTREE algorithm, where the tree is represented by θ. The length of history recorded is given by the maximum depth of the tree.

A similar algorithm to UTREE was described by Dutech [2000], with an enhancement for cases when a deterministic POMDP model is known. The enhancement grows the UTREE when an examination of the POMDP model determines that a path down the tree still results in an ambiguous world-state, and therefore a possibly ambiguous action.

Deterministically assigning actions to history tree leaves can lead to large trees and over-fitting in noisy worlds. Suematsu and Hayashi [1999] uses Bayesian inference to construct future reward and observation models as a function of a learnt history tree, providing a smoother model than UTREE. The policy can be constructed by performing DP on the history tree model, similar to performing DP on HMMs.

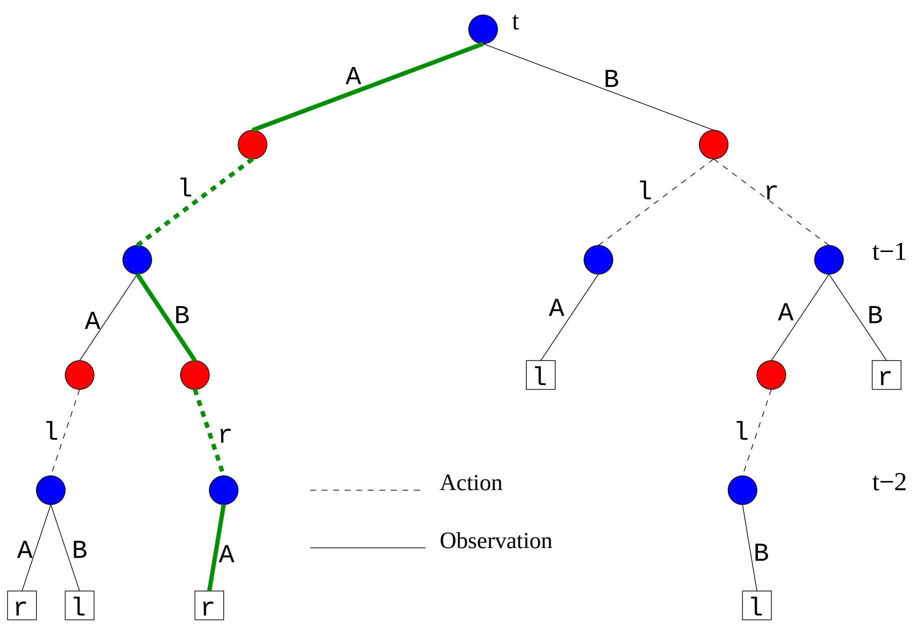


Figure 2.6: A history tree learnt by UTREE for some POMDP with $\mathcal{Y} = \{\mathtt{A},\ \mathtt{B}\}$ and $\mathcal{U} = \{\mathtt{l},\ \mathtt{r}\}$. Leaves are labelled with optimal actions and a sample path is shown for the history $\{\mathtt{(A,r)(B,l)(A,\cdot)}\}$ choosing action $\mathtt{r}$. The most recent observation gives the first branch to take from the root of the tree.

2.6.2.3 RNN Based Methods

Recurrent neural networks (RNNs) augment the output of an ANN with continuous state outputs that are fed back into a previous layer of the network (see Figure 2.7). The network has a continuous internal-state space allowing a POMDP to be controlled by presenting observations as inputs and interpreting outputs as action distributions (Recurrent-model [Lin and Mitchell, 1992]), Q-values (Recurrent-Q [Lin and Mitchell, 1992]), or by learning a second function mapping the internal state and future input predictions to actions [Schmidhuber, 1991].

The agents in this section learn both ϕ: the recurrent state model, and θ: the Q-value or action outputs. Here $\mathcal{G}$ is the set of vectors in $\mathbb{R}^n$ that are the n RNN outputs fed back to the previous layer. The internal-state update is deterministic and the actions may be chosen deterministically or stochastically.

One RNN training algorithm is back-propagation through time (BPTT) [Rumelhart et al., 1986, §8] that unfolds the network over T steps then uses the standard error back propagation algorithm for ANNs. This means that the network can only be explicitly trained to have a T step memory. However, during operation the internal state may carry relevant information for more than T steps. Real-time recurrent learning (RTRL) [Williams and Zipser, 1989] overcomes this problem by efficiently propagating errors

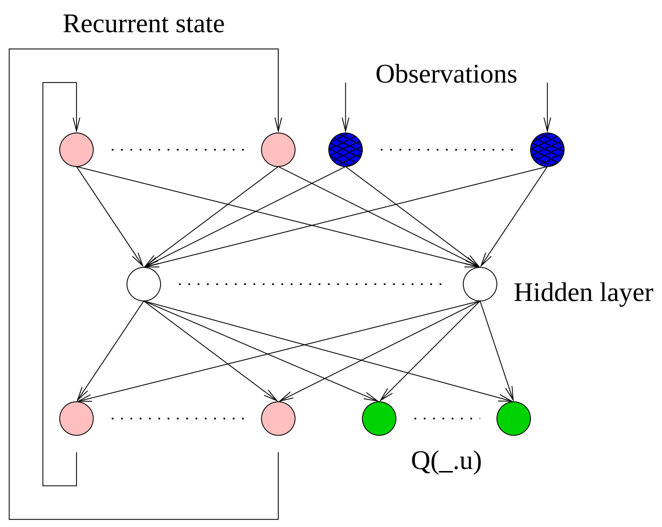


Figure 2.7: A simple recurrent neural network for learning Q-values.

back to time 0. These algorithms have been applied to speech recognition [Bourlard and Morgan, 1994], hinting that speech processing can be studied in a POMDP framework. RNNs are more throughly introduced in Appendix E.4.3.

Unfortunately BPTT and RTRL have difficulty learning long-term memory because the back propagated error signals tend to blow up or vanish depending on the feedback weights. Learning long term memory appears to be a problem for all model-free POMDP algorithms. History compression [Schmidhuber, 1992] uses an RNN to filter out short term information by attempting to predict its own next input as well as learning actions. When the RNN prediction fails the input is passed to a higher level RNN that concentrates on modelling long-term features of the input.

Alternatively, long short-term memory (LSTM) introduces complex neurons that learn to turn on or off their memory inputs and outputs. Another feature of LSTM is that the back-propagated error signal does not explode or vanish [Hochreiter and Schmidhuber, 1997]. LSTM has been applied to navigation POMDPs constructed to require up to 70 steps of memory [Bakker, 2001]. The LSTM network learns Q-values and uses the interesting idea of adapting the Q-learning exploration probability by training a non-recurrent ANN to predict the variance of the Q-values.

2.6.2.4 Evolutionary Methods

Evolutionary methods create populations of agents and use combination and mutation rules to generate new populations. The long-term average reward is the natural fitness measure of an agent. Methods include rule based agents [Kwee et al., 2001], lookup-table agents [Moriarty et al., 1999], and probabilistic program agents [Sałustowicz and Schmidhuber, 1997]. The last two both have the advantage of allowing stochastic agents. There is a strong connection between reactive Q-learning and some evolutionary classifier algorithms [Lanzi, 1997].

Evolutionary algorithms can resolve partial observability by adding memory registers that the agent can learn to set. One successful example is a POMDP algorithm that evolves stochastic RNNs to control large POMDPs such as the New York driving scenario [Glickman and Sycara, 2001]. The hidden units of the RNN each choose an output of 1 or 0 from a distribution controlled by the squashed sum of the weighted inputs. A single output unit causes one of 3 actions to be chosen depending on which of 3 ranges the output value falls in. In this case ϕ represents the weights of the hidden units, and θ represents the weights of the output unit. We will discuss this method in more detail in Section 7.1 since there is a close relationship between Glickman's approach, the approach of this thesis, and the methods discussed in the next section.

2.6.2.5 Finite State Controllers

The idea behind finite state controllers (FSCs) is that past events which are relevant to choosing optimal actions can be remembered indefinitely by a directed cyclic graph of internal states. Each node of the FSC is an I-state from $\mathcal{G}$. This model uses ϕ to parameterise probabilities of I-state transitions based on the current I-state and observation. The next state is chosen stochastically from the distribution $\omega(\cdot|\phi, g, y)$. Similarly, θ parameterises learnt action probabilities for each I-state, so that u is chosen stochastically from $\mu(\cdot|\theta, h, y)$. This is the internal-state model we adopt for the algorithms in this thesis so we devote some time to motivating it.

The agent learns to use the I-states to remember only what is needed in order to act optimally. This process can be viewed as an automatic quantisation of the belief state space to provide the optimal policy *representable by* $|\mathcal{G}|$ *I-states.* As $|\mathcal{G}| \to \infty$ we can represent the optimal policy arbitrarily accurately [Bonet, 2002].

Another way to view this process is as direct learning of a stochastic *policy graph* [Meuleau et al., 1999b]. Recall from Section 2.5.1.3 that the nodes of a policy graph are synonymous with the I-states of an FSC. However, exact algorithms compute policy graphs with deterministic I-state transitions equivalent to a deterministic $\omega(h|\phi, g, y)$ function. Furthermore, they permit only *one action per node.* We allow stochastic I-state transitions and the policy $\mu(u|\theta, h, y)$ allows a different action distribution for each I-state *and each observation.* This means we can compute optimal policies with far fewer I-states than the equivalent policy graph. We will give a concrete example when investigating the Heaven/Hell scenario in Section 8.2.

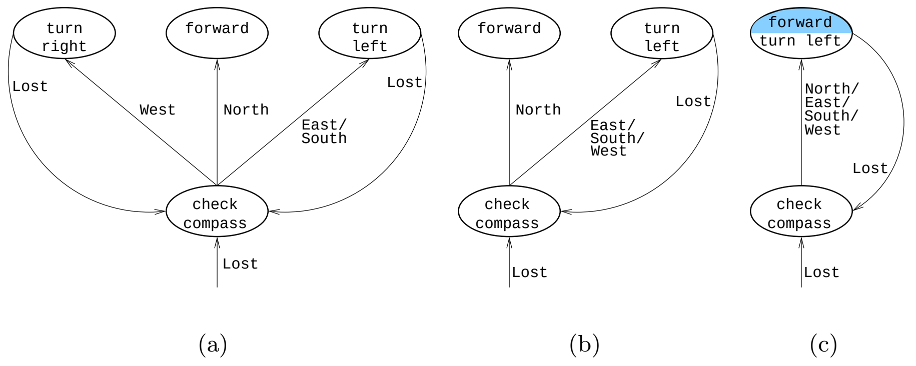


Figure 2.8: Figure 2.8(a) shows the optimal policy graph fragment for a lost agent that must move north. If the number of I-states is reduced to 3 (2.8(b)) then the policy is still deterministic but degraded because it may turn left 3 times instead of right once. In Figure 2.8(c) the agent cannot determine if it is facing north when it is in the upper I-state, thus must sometimes move forward and sometimes turn.

A third way to view the process is as a grid method (recall Section 2.5.2.2) where the number of points is fixed, but the location of the points can move around to best cover the interesting regions of belief state space.

Though it is a small comfort, training an agent from the class of algorithms based on restricted FSCs has NP-hard complexity instead of the PSPACE-hard complexity of exact algorithms [Meuleau et al., 1999a].

The FSC framework encompasses other algorithms that differ by how they learn $\omega(h|\phi, g, y)$. For example, Peshkin et al. [1999] and Peshkin [2000], where the action space of SARSA or VAPS is augmented with external-memory-setting actions. Also, the algorithms of Sections 2.6.2.1–2.6.2.4 can be cast as FSC methods. FSCs can also be learnt by depth first search in constrained spaces of FSC [Meuleau et al., 1999b].

Section 2.6.1 discusses the necessity of stochastic policies for POMDPs in the absence of memory. Stochasticity is also needed when $|\mathcal{G}|$ is too small. When there is insufficient memory to resolve the uncertainty about the current state there is insufficient information for determining which action is best. Consider Figure 2.8, showing a fragment of an FSC in which a lost agent uses a compass to face north before moving on. As the number of I-states is reduced the policy degrades until for $|\mathcal{G}| = 2$ the agent cannot decide between moving forward or turning, which is a stochastic policy. Only finitely transient POMDPs (see Section 2.5.1.3) have optimal deterministic FSCs, that is, an optimal policy that can be represented by a policy graph.

Finite state controllers have been proposed as a useful method for fully observable MDPs with very large state spaces where the optimal reactive policy is too hard to represent. Kim et al. [2000] uses VAPS to perform policy-search in the space of FSCs where the state is fully observable but factored as described in Section 2.5.3.

Finite state controllers exhibit difficulties when learning non-trivial I-state functions such as those requiring long-term memory. When all I-state trajectories are nearly equally likely to occur for any world-state trajectory, there can be little correlation between a particular I-state trajectory and a high reward, hence no single I-state trajectory is reinforced. This explanation for difficulties with long term memory was proposed for value function methods by Lanzi [2000] and termed *aliasing on the payoffs*. We analyse this problem for policy-gradient methods in Chapter 7, and propose a solution that worked for the experiments reported in this thesis.

2.6.3 Policy-Gradient Methods

This section introduces policy-gradient methods for model-free agent training — one of the approaches we take in this thesis. Policy gradient methods compute (or estimate) the gradient of $\eta(\phi, \theta)$ (2.1) with respect to the parameters of the agent ϕ and θ. This allows η to be maximised by some form of gradient ascent procedure. In Section 2.5.7 we discussed some advantages of using policy-gradient methods instead of value-function methods. Another advantage is that gradient ascent is a local optimisation method. This means means we can avoid the computation complexity of exact POMDP methods. However, the trade-off is that we cannot guarantee convergence to the global maximum of $\eta(\phi, \theta)$. Consideration must also be given to the conditions under which the gradient is well defined. In Chapter 3 we will make mild assumptions about the world and the agents to ensure the gradient exists. Broadly speaking, we assume the world-state Markov process is ergodic, and the processes $\omega(h|\phi, g, y)$ and $\mu(u|\theta, h, y)$ are differentiable.

It is easy to add domain knowledge to agents in the policy-gradient setting. Hard-wired rules can modify the distributions $\omega(h|\phi, g, y)$ and $\mu(u|\theta, h, y)$ additively, hence having no impact on the gradient computation. Domain knowledge can also be used to shape the rewards [Laud and DeJong, 2002], or to factor the observations.

Early algorithms that estimated gradients of performance measures using Monte-Carlo like methods include the *likelihood-ratio* method [Aleksandrov et al., 1968, Rubinstein, 1969]. Extensions of the likelihood-ratio method to *regenerative processes* (including Markov Decision Processes) were given by Glynn [1986, 1990], Glynn and L'Ecuyer [1995], Reiman and Weiss [1986], and Reiman and Weiss [1989]; and independently for finite-horizon MDPs by Williams [1992]. Glynn showed that the gradient could be estimated by sampling a trajectory through the state space for a T step episode of an finite-horizon MDP. The estimated gradient extended to the POMDP setting is given by

$$\widehat{\nabla\eta} = \left(\sum_{t=1}^{T} r_t\right) \sum_{t=1}^{T} \frac{\nabla\mu(u_t|\theta, y_t)}{\mu(u_t|\theta, y_t)}, \tag{2.10}$$

where the gradient is with respect to the policy parameters θ. This is a memory-less agent which assumes that y_t reveals sufficient world-state to act well. This scheme requires $\frac{\nabla\mu(u_t|\theta,y_t)}{\mu(u_t|\theta,y_t)}$ be well behaved. This is the case if $\mu(u_t|\theta, y_t)$ is parameterised so that $\nabla\mu(u_t|\theta, y_t)$ and $\mu(u_t|\theta, y_t)$ go to zero at the same rate. We discuss one such choice of $\mu(u_t|\theta, y_t)$, the soft-max function, in Section 3.2. Once the gradient has been estimated a stochastic gradient ascent algorithm can be used to maximise η. Equation (2.10) does not take into account the causality of rewards, that is, rewards gathered *before* time t can contribute to the gradient induced by actions *after* time t. A simple fix to this is implemented by Meuleau et al. [2001], replacing the sum of rewards with the sum of rewards up to time t.

Williams' REINFORCE is unbiased. It converges to the true gradient over many episodes. Williams' REINFORCE only works for episodic tasks because the algorithm awards *equal credit* for the reward up to time t to each action. If we attempted to use Williams' REINFORCE on an infinite-horizon POMDP, using a single infinitely-long sample trajectory through state space, we cannot assign credit to specific actions because there are infinitely many of them. In order to apply Williams' REINFORCE to POMDPs it is necessary to be able to identify when the system enters a *recurrent state*, indicating the end of an episode and bounding the implications of actions on the long-term reward. This is non-trivial when the true state is masked by the observation process $\nu(y|i)$.

If the agent cannot observe visits to recurrent states, or if visits occur infrequently, we resort to biased estimates of the gradient. A number of researchers introduced discounted eligibility traces [Marbach and Tsitsiklis, 2000, Kimura et al., 1997] to bound the implications of actions, but the discount factor introduces a bias. The eligibility trace acts as memory, telling the agent what proportion of the current reward each previous action should be credited with.

The similar approach of Baxter and Bartlett [2001] is to derive, from scratch, an algorithm that directly approximates $\nabla\eta$ for infinite-horizon POMDPs. This derivation also introduces a discounted eligibility trace with discount factor $\beta \in [0, 1)$, showing that in the limit as $\beta \to 1$ the approximated gradient converges to $\nabla\eta$. The algorithm is summarised by the following expression that can be unrolled to form the core of the GPOMDP algorithm [Baxter and Bartlett, 2000, Barlett and Baxter, 2000, Baxter and

Bartlett, 2001]

$$\widehat{\nabla\eta} = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \frac{\nabla\mu(u_t|\theta, y_t)}{\mu(u_t|\theta, y_t)} \sum_{s=t+1}^{T} \beta^{s-t-1} r_s.$$

The left hand side is the approximation of the true gradient. As $\beta \to 1$ the approximation becomes perfect, but the variance goes to ∞. When $\beta < 1$ we make the assumption that the effect of actions on the reward decays exponentially with time, allowing the temporal credit assignment problem to be solved. Other methods of bounding the period in which rewards can be linked to actions are valid but the exponential decay model is easy to analyse and implement.[4] Baxter et al. [2001a] shows that the variance of the estimate scales with $1/\left[T(1-\beta)\right]$, which reflects the fact that as the credit for rewards is spread over a longer time, the temporal credit assignment problem becomes harder, until the case of $\beta = 1$ gives us no discounting and the variance is infinite. The GPOMDP algorithm was proved to extend to the setting of infinite state and action spaces. An example of a continuous action space is given by Kimura and Kobayashi [1998]. Convergence *rates* for GPOMDP in terms of the POMDP *mixing time* have also been established [Bartlett et al., 2000, Bartlett and Baxter, 2002].

Williams' REINFORCE has been used as a policy-gradient algorithm for learning FSCs [Peshkin, 2000] but in its basic form is restricted to POMDPs for which a recurrent state can be identified. One of the contributions of this thesis is to extend GPOMDP to learning FSCs, allowing policy-gradient methods to learn FSCs for controlling *infinite-horizon* POMDPs.

2.7 Further Issues

This section presents useful ideas for improving most POMDPs methods.

2.7.1 Variance Reduction for POMDP Methods

Monte-Carlo algorithms for POMDPs have high-variance, policy-gradient methods in particular. This section describes established methods for reducing the variance.

2.7.1.1 Importance Sampling

Importance Sampling (IS) is a method for improving estimates of functions of arbitrary distributions that are known only up to a normalising constant. The idea is to sample from an appropriately selected *known* distribution. Samples are weighted by the ratio of the probability under their true distribution to their probability under the sampling distribution. The sampling distribution should be our best guess of the true

[4]This is true of a broad class of reinforcement learning algorithms that use discount factors.

distribution. This means the weights reflect the relative importance of each sample of the unknown distribution. Variance reduction occurs when more weight is given to the areas of the sampling distribution where the bulk of the unknown distribution's probability mass is concentrated.

The application of IS to Monte-Carlo methods is discussed by Glynn [1996] and applied to Williams' REINFORCE and VAPS by Meuleau et al. [2000]. IS can be used to implement *off-policy* policy-gradient methods [Peshkin and Shelton, 2002], reducing the number of world interactions required as described in Section 2.4. These methods permit computation of bounds on the probability of finding an ϵ-approximation of the true value using a finite number of samples from the world [Peshkin and Shelton, 2002, Peshkin, 2002]. IS and Williams' REINFORCE have been used in conjunction with learning finite state controllers (Section 2.6.2.5) as well as reactive policies [Shelton, 2001c,b]. It should be possible to use IS with the methods described in this thesis, but Section 9.3 shows that the same method that can be applied to Williams' REINFORCE cannot be naively applied to our algorithms unless the stationary distribution over states is known.

IS has also been extended to monitoring belief states with transitions modeled by BNs, directly minimising the variance of the belief state estimates, combining the ideas of Sections 2.5.3.1 and 2.5.6 [Ortiz and Kaelbling, 2000].

2.7.1.2 Reward Baselines

Marbach and Tsitsiklis [1999] use $r_t - \eta$ as a performance indicator at each time step instead of just the long-term reward η. The difference between r_t and the reward *baseline* η indicates whether the reward was above or below average. Intuitively, this indicates whether positive or negative reinforcement is required, providing variance reducing information. Weaver and Tao [2001] proved that η is the optimal baseline for policy-gradient algorithms. Greensmith et al. [2002] show that the widely used J_β baseline is *sub-optimal*. That paper also analyses the use of general additive control variates, such as actor-critic methods, to reduce variance. These ideas are similar to the actor-critic style estimates proposed by Sutton et al. [2000] and Konda and Tsitsiklis [2000]. These methods can be applied immediately to the algorithms in this thesis though we have not yet investigated the effect of doing so.

2.7.1.3 Fixed Random Number Generators

A source of variance in value-function or policy-gradient estimates arises from uncertainty in the transitions, that is, it is possible to execute T policy steps starting from the same start state and obtain different average rewards. Ng and Jordan [2000] extend the work of Kearns et al. [1999], developing the PEGASUS algorithm that is applicable to POMDPs simulated with a controlled source of randomness. It transforms arbitrary POMDPs into POMDPs with deterministic transitions by augmenting the state space with a record of the random numbers that will be used to generate the state and observation trajectories. Long-term rewards are then estimated starting from m initial states drawn at random. The m needed to obtain ϵ-convergence of the value-function scales polynomially with the discount factor and the complexity of the policy space, but not with the number of states in the POMDP. In practice this method can be applied by re-seeding the simulator's random number generator to a fixed value before each episode. PEGASUS is similar to *paired* statistical tests for policy evaluation and is compared to them under various optimisation methods by Strens and Moore [2001].

We apply the method to our Monte-Carlo policy-gradient algorithms, reducing the variance in gradient estimates. The variance reduction is not free since the method can introduce false local maxima. We are not aware of any literature discussing this phenomenon so Section 9.4 describes it.

2.7.2 Multi-Agent Problems

All the algorithms discussed in this thesis can be easily extended to multiple agents. The idea is to alter the set of actions $\mathcal{U}$ such that it contains the cross product of all the actions available to each agent, that is, for n agents, $\mathcal{U} = \{\mathcal{U}_1 \times \mathcal{U}_2 \times \cdots \times \mathcal{U}_n\}$. If the agent parameters are independent, then each agent independently chooses actions that are combined to form the meta-action. For stochastic policies, the overall action distribution is just the joint distribution of actions for each agent, $\mu(u_1, u_2, \dots | \theta_1, \theta_2, \dots, h_1, h_2, \dots, y_1, y_2, \dots)$.

Examples of multi-agent learning in the policy-gradient setting include training independent neurons in the brain [Barlett and Baxter, 1999] and training multiple network routers for maximum throughput [Tao et al., 2001]. Dutech et al. [2001] uses an on-line version of the GPOMDP algorithm to incrementally train more and more agents, in larger and larger worlds. By using value functions, multiple automated guided vehicles have been trained to operate in a factory [Makar et al., 2001].

We investigate one multi-agent problem in Section 4.5, showing how our model-based algorithm can train two robots to interact on a factory floor without explicit communication.

2.8 Summary

Table 2.1 summarises the model-based algorithms described in this chapter. Table 2.2 correspondingly summarises the model-free algorithms. Particular attention is paid to comparing the how the $\omega(h|\phi, g, y)$ and $\mu(u|\theta, h, y)$ processes are parameterised for each algorithm.

Key Points

⊶ I POMDP methods can be split along several axes:

- model-based learning or model-free learning;
- exact or approximate agents;
- policies inferred from values or learnt directly;
- policies computed analytically or via simulation.

⊶ II Exact methods are PSPACE-hard so we need tractable approximate methods.

⊶ III POMDP agents need memory to act optimally.

⊶ IV Policy-gradient methods do not waste effort learning and representing values that are discarded after the policy is formed.

⊶ V Policy-gradient methods guarantee convergence to at least a local maximum, but they may take longer to do so than value methods due to high variance in the gradient estimates.

Some fields of POMDP research have been left out of this discussion because they are not pertinent to this thesis. They include:

- Frameworks that sit between MDPs and POMDPs [Pendrith and McGarity, 1998, Zubeck and Dietterich, 2000];
- Hierarchical POMDPs [Dietterich, 2000, Theocharous et al., 2000, Hernandez-Gardio and Mahadevan, 2001, Thiébaux et al., 2002];
- Multiple time scale and continuous time POMDPs [Precup, 2000, Ghavamzadeh and Mahadevan, 2001].

The remainder of this thesis is only concerned with policy-gradient methods where memory is provided by FSCs or HMMs.

Table 2.1: A summary of the model-based algorithms described in this chapter. The ω column contains `d` if the internal state update is deterministic and `s` if the update is stochastic. Similarly, the μ column indicates if the choice of action is deterministic of stochastic. Uppercase `D` indicates that the function is fixed instead of learnt. The last column describes how ϕ parameterises $\omega(h|\phi, g, y)$ and θ parameterises $\mu(u|\theta, h, y)$.

Method	ω	μ	ϕ *parameters*	θ *parameters*
MDP [§2.4] e.g. Value Iteration	d	d	Fully observable, therefore no memory required	θ stores long-term value of each state
Exact [§2.5.1] e.g. Incremental Pruning	D	d	I-state is b_t, ϕ represented by $q(j|i,u)$ and $\nu(y|i)$	Piecewise linear hyperplanes or policy graph
Heuristic [§2.5.2.1]	D	d	As for exact	Approximation to piecewise linear hyperplanes
Grid [§2.5.2.2]	D	d	As for exact	$\mu(u|\theta, h, y)$ interpolates grid point values stored by θ
Factored belief [§2.5.3.1]	d	d	ϕ encodes *Bayesian network* or *algebraic decision tree*	Could be any of the previous parameterisations
Factored value [§2.5.3.2]	d	d	Any of previous, including exact and factored	Linear combinations of basis functions or ADDs
Planning [§2.5.4]	d	D	Any belief state tracking method	$\mu(u|b, y)$ searches space of future trajectories
RTDP-Bel [§2.5.5]	d	d	Any method to track sampled beliefs b_t	θ stores value of all visited belief states
SPOVA & Linear-Q [§2.5.5]	d	d	Any method to track sampled beliefs b_t	Smooth approx. of exact $\bar{J}_\beta$ learnt by gradient ascent
Particle Filters [§2.5.6]	d	d	ϕ represents tracked b_t as n particles	KNN to infer value for b_t represented by particles
Policy Iteration [§2.5.7]	d	d	ϕ is a policy graph (PG) converted to $\bar{J}_\beta$ for learning steps	θ maps policy graph nodes to actions, learnt during DP
Depth first PG search [§2.5.7]	d	d	ϕ chosen by search of a tree of constrained PGs	θ maps PG nodes to actions
Gradient ascent of PGs [§2.5.7]	s	s	ϕ is PG transition probs. learnt by gradient ascent	θ stochastically maps PG nodes to actions
Approx. Gradient PG [§2.5.7]	s	s	PG transition probabilities controlled by ANN	ANN maps PG nodes to action probabilities
Actor-Critic & VAPS[§ 2.5.8]	D	s	Usually no internal state	θ is policy learnt by gradient ascent *and* value-iteration

Table 2.2: A summary of the model-free algorithms described in this chapter. Each column has the same meaning as Table 2.1. The tables are not a complete summary of all POMDP algorithms.

Method	ω	μ	ϕ *parameters* / θ *parameters*
JSJ Q-learning [§2.6.1]	D	s	Assume $y_t = i_t$, so no internal state
			θ stores long-term values of each observation
HMM methods [§2.6.2.1]	d	s	ϕ is HMM transition probabilities
			θ is action probs. or value of HMM belief states
Window-Q [§2.6.2.2]	D	d	ϕ deterministically records last n observations $\bar{y}$
			θ is ANN weights mapping $\bar{y}$ to values
UTREE [§2.6.2.2]	D	d	ϕ deterministically records last n observations $\bar{y}$
			θ represents tree; follow $\bar{y}$ branch to get u_t
RNNs [§2.6.2.3]	d	d	RNN maps y_t & RNN state output to new state output
			RNN maps y_t & RNN state output to actions or values
Evolutionary [§2.6.2.4]	s	D	ϕ is RNN trained using EAs & stochastic sigmoid outputs
			θ weights sigmoid outputs to select actions
FSCs [§2.6.2.5]	s	s	ϕ is prob. of I-state transition $g \rightarrow h$
			θ is prob. of u_t given I-state h
Williams' REINFORCE [§2.6.3]	s	s	I-states may be changed by memory setting actions
			Grad ascent of θ that maps $y_t \rightarrow u_t$
GPOMDP [§2.6.3]	s	s	Learning ϕ is the subject of Chapter 5
			Grad ascent of θ that maps $y_t \rightarrow u_t$ for infinite-horizons

Chapter 3

Stochastic Gradient Ascent of FSCs

He had bought a large map representing the sea,
Without the least vestige of land:
And the crew were much pleased when they found it to be
A map they could all understand.
—Charles Lutwidge Dodgson

Our aim is to maximise $\eta(\phi, \theta, i, g)$, the long-term average reward (2.1), by adjusting the parameters of the agent in the direction of the gradient $\nabla\eta(\phi, \theta, i, g)$. Before Chapters 4–6 describe several algorithms for doing this, we use this chapter to state the key quantities and assumptions we rely on to ensure the existence of $\nabla\eta(\phi, \theta, i, g)$. Firstly, we show how to construct a single Markov chain from the world-state, the I-state in the form of a finite state controller (FSC), and the policy. Then we show how to construct the functions $\omega(h|\phi, g, y)$ and $\mu(u|\theta, h, y)$ to represent an FSC and policy such that the necessary assumptions are satisfied. This is achieved using the *soft-max* function that generates distributions from the real valued output of a function approximator. The soft-max function is used for all the experiments documented in this thesis. We also briefly describe our conjugate gradient ascent procedure with details deferred to Appendix B.

3.1 The Global-State Markov Chain

Recall from Section 2.1 that the transition probabilities governing the world-state Markov process are described by $q(j|i, u)$. Similarly, the transition probabilities between I-states are described by the FSC transition probabilities $\omega(h|\phi, g, y)$. From Meuleau et al. [1999a, Thm.1], the evolution of global states (world-state and I-state pairs (i, g)) is also Markov, with an $|\mathcal{S}||\mathcal{G}| \times |\mathcal{S}||\mathcal{G}|$ transition probability matrix $P(\phi, \theta)$. The entry in row (i, g) and column (j, h) is given by

$$p(j, h|\phi, \theta, i, g) = \sum_{y \in \mathcal{Y}} \sum_{u \in \mathcal{U}} \nu(y|i)\omega(h|\phi, g, y)\mu(u|\theta, h, y)q(j|i, u). \tag{3.1}$$

This equation computes the expectation over all observations and actions of global state transition $(i,g) \to (j,h)$. The model of world given by $\nu(y|i)$ and $q(j|i,u)$ must be known before $P(\phi,\theta)$ can be computed explicitly.

A step in computing $\nabla\eta(\phi,\theta,i,g)$ needs the gradient of the global-state transition matrix

$$\begin{aligned}\nabla P &= [\nabla^{\phi} P, \nabla^{\theta} P] &(3.2)\\ &= \left[\frac{\partial P}{\partial \phi_1}, \ldots, \frac{\partial P}{\partial \phi_{n_\phi}}, \frac{\partial P}{\partial \theta_1}, \ldots, \frac{\partial P}{\partial \theta_{n_\theta}}\right]. &(3.3)\end{aligned}$$

The partial derivative of the matrix P with respect to parameter ϕ_l where $l \in \{1,\ldots,n_\phi\}$ is the element-wise derivative

$$\frac{\partial P}{\partial \phi_l} = \begin{bmatrix} \frac{\partial p(1,1|\phi,\theta,1,1)}{\partial \phi_l} & \cdots & \frac{\partial p(|\mathcal{S}|,|\mathcal{G}||\phi,\theta,1,1)}{\partial \phi_l} \\ \vdots & \ddots & \vdots \\ \frac{\partial p(1,1|\phi,\theta,|\mathcal{S}|,|\mathcal{G}|)}{\partial \phi_l} & \cdots & \frac{\partial p(|\mathcal{S}|,|\mathcal{G}||\phi,\theta,|\mathcal{S}|,|\mathcal{G}|)}{\partial \phi_l} \end{bmatrix}.$$

Each element of each $\frac{\partial P}{\partial \phi_l}$ is given by

$$\frac{\partial p(j,h|\phi,\theta,i,g)}{\partial \phi_l} = \sum_{y,u} \nu(y|i) \frac{\partial \omega(h|\phi,g,y)}{\partial \phi_l} \mu(u|\theta,h,y) q(j|i,u). \tag{3.4}$$

The corresponding entries for $\frac{\partial P}{\partial \theta_c}$, where $c \in \{1,\ldots,n_\theta\}$, are

$$\frac{\partial p(j,h|\phi,\theta,i,g)}{\partial \theta_c} = \sum_{y,u} \nu(y|i) \omega(h|\phi,g,y) \frac{\partial \mu(u|\theta,h,y)}{\partial \theta_c} q(j|i,u). \tag{3.5}$$

3.2 Conditions for the Existence of $\nabla\eta$

As we did for P, we drop the explicit dependence of the long-term average reward, $\eta(\theta,\phi,i,g)$, on the parameters θ and ϕ. We shall see in this section that, under our assumptions, η is also independent of i and g. Thus, η is a scalar that depends implicitly on only the parameters θ and ϕ. The gradient of η has $n_\phi + n_\theta$ components

$$\begin{aligned}\nabla\eta &= [\nabla^{\phi}\eta, \nabla^{\theta}\eta] \\ &= \left[\frac{\partial \eta}{\partial \phi_1}, \ldots, \frac{\partial \eta}{\partial \phi_{n_\phi}}, \frac{\partial \eta}{\partial \theta_1}, \ldots, \frac{\partial \eta}{\partial \theta_{n_\theta}}\right].\end{aligned}$$

Section 4.1 describes how to compute $\nabla\eta$, however, to ensure $\nabla\eta$ is well defined we assume the following conditions. As we shall discuss, the conditions are either mild or surmountable.

Assumption 1. *Each* $P(\phi,\theta)$ *has a unique stationary distribution*

$$\pi(\phi,\theta) := [\pi(1,1|\phi,\theta),\dots,\pi(|\mathcal{S}|,|\mathcal{G}||\phi,\theta)]',$$

satisfying the balance equations. *This implies that* $\pi(\phi,\theta)$ *is the leading left eigenvector of* $P(\phi,\theta)$. *We use* $'$ *to denote the transpose operator.*

$$\pi'(\phi,\theta)P(\phi,\theta) = \pi'(\phi,\theta). \tag{3.6}$$

Assumption 2. *The rewards are uniformly bounded by* $|r(i)| < R < \infty \ \forall i \in \mathcal{S}$.

Assumption 3. *The derivatives,* $\frac{\partial\omega(h|\phi,g,y)}{\partial\phi_l}$ *and* $\frac{\partial\mu(u|\theta,h,y)}{\partial\theta_c}$ *are uniformly bounded by* $Q < \infty$ *and* $U < \infty$ *respectively* $\forall g,h \in \mathcal{G}$, $u \in \mathcal{U}$, $y \in \mathcal{Y}$, $\theta_c \in \mathbb{R}$, *and* $\phi_l \in \mathbb{R}$.

Assumption 4. *The ratios*

$$\frac{\left|\frac{\partial\omega(h|\phi,g,y)}{\partial\phi_l}\right|}{\omega(h|\phi,g,y)} \quad \textit{and} \quad \frac{\left|\frac{\partial\mu(u|\theta,h,y)}{\partial\theta_c}\right|}{\mu(u|\theta,h,y)}$$

are uniformly bounded by $B < \infty$ *and* $D < \infty$ *respectively* $\forall g,h \in \mathcal{G}$, $u \in \mathcal{U}$, $y \in \mathcal{Y}$, $\phi_l \in \mathbb{R}$, *and* $\theta_c \in \mathbb{R}$.

Assumption 1 ensures that the Markov chain generated by $P(\phi,\theta)$ has a unique recurrent class (an ergodic Markov chain), which mainly just makes the statement of the theorems more compact. In the non-unique case the theorems can be generalised by considering each recurrent class of global states independently. Alternatively, ergodicity can be guaranteed by modifying P to include small fixed probabilities of transitions to random states.

Recall Equation (2.1), the expression for $\eta(\phi,\theta,i,g)$. We now show that under Assumption 1, $\eta(\phi,\theta,i_0,g_0)$ is independent of the starting state (i_0,g_0) and is equal to

$$\begin{aligned}\eta(\phi,\theta) &= \pi'(\phi,\theta)r \qquad (3.7)\\ &= \sum_{i\in\mathcal{S}}\sum_{g\in\mathcal{G}}\pi(\phi,\theta,i,g)r(i)\end{aligned}$$

where $r := [r(1,1),\dots,r(|\mathcal{S}|,|\mathcal{G}|)]'$ and $r(i,g) := r(i) \ \forall g \in \mathcal{G}$. To demonstrate that η is independent of the starting state we use the Ergodic Theorem to show that with probability 1 η equals the expected reward over an infinite sample of the global-state Markov process $\{S_t, G_t\} \sim \pi(\phi,\theta)$. Recall that $\chi_i(S_t)$ is the indicator function (2.6), which is 1 when $S_t = i$ and 0 otherwise, thus

$$\begin{aligned}\sum_{i,g} \pi(\phi,\theta,i,g) r(i) &= \sum_{i,g} \mathbb{E}_{\phi,\theta} \chi_i(S_t)\chi_g(G_t) r(S_t) \\ &= \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} \sum_{i,g} \chi_i(S_t)\chi_g(G_t) r(S_t) \text{ w.p.1} \\ &= \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} r(S_t) \text{ w.p.1},\end{aligned}$$

where the Ergodic Theorem takes us from the first line to the second. The result is true for all sample sequences $\{(i_0, g_0), (i_1, g_1), \ldots, (i_T, g_T)\}$ except those with measure zero. Because the rewards are bounded by Assumption 2 we can apply Lebesgue's Dominated Convergence Theorem to take the expectation over all sample sequences to obtain (2.1)

$$\eta(\phi,\theta,i,g) := \lim_{T\to\infty} \frac{1}{T} \mathbb{E}_{\phi,\theta} \left[\sum_{t=1}^{T} r(i_t) | i_0 = i, g_0 = g \right].$$

After this chapter will usually drop the explicit dependence on ϕ, θ for brevity, however, it is always implied.

3.3 Generating Distributions with the Soft-Max Function

Assumption 4 might seem to prevent deterministic policies. Fortunately, some choices of $\mu(u|\theta, h, y)$ and $\omega(h|\phi, g, y)$ satisfy Assumptions 3 and 4 while still allowing deterministic policies, provided we make the appropriate limiting definition of $\frac{0}{0}$. For example, any function that uses a soft-max distribution based on underlying real parameters will satisfy these conditions, as will many others. Thus, the soft-max function is a useful distribution which becomes deterministic in the limit as the real parameters diverge to infinity.

Definition 2. *Given a vector $\rho \in \mathbb{R}^n$, the probability of event $m \in \{1, \ldots, n\}$ according to the soft-max distribution generated by ρ is*

$$\Pr(m) := \frac{\exp(\rho_m)}{\sum_{m'=1}^{n} \exp(\rho_{m'})}. \tag{3.8}$$

$$\frac{\partial \Pr(m)}{\partial \rho_c} = \Pr(m)(\chi_c(m) - \Pr(c)). \tag{3.9}$$

We typically compute the derivative of the soft-max function with respect to the real parameters ϕ_l or θ_c after sampling an event m from the soft-max generated distribution.

3.4 Agent Parameterisations

Except for the LVCSR speech recognition experiments, we restrict ourselves to the following two forms of agent parameterisation.

3.4.1 Lookup Tables

Unless otherwise stated, all experiments use FSCs and policies parameterised by tables indexed by the current I-state and observation, (g, y). Each index provides a vector of $|\mathcal{G}|$ or $|\mathcal{U}|$ real parameters for $\omega(h|\phi, g, y)$ and $\mu(u|\theta, h, y)$ respectively. Thus, each parameter can be labelled with the index (I-state g and observation y), and either the next I-state h or action u that it represents. For example, the parameter ϕ_{gyh} is a real number that controls the relative probability of making the I-state transition $g \rightarrow h$ having observed y. The *soft-max* function is used to generate distributions from the indexed vector. Applying Equation (3.8) gives us

$$\begin{aligned}
\omega(h|\phi, g, y) &= \frac{\exp(\phi_{gyh})}{\sum_{h' \in \mathcal{G}} \exp(\phi_{gyh'})} \\
\mu(u|\theta, h, y) &= \frac{\exp(\theta_{hyu})}{\sum_{u' \in \mathcal{U}} \exp(\theta_{hyu'})}.
\end{aligned}$$

Applying Equation (3.9), gives us the gradient ratios

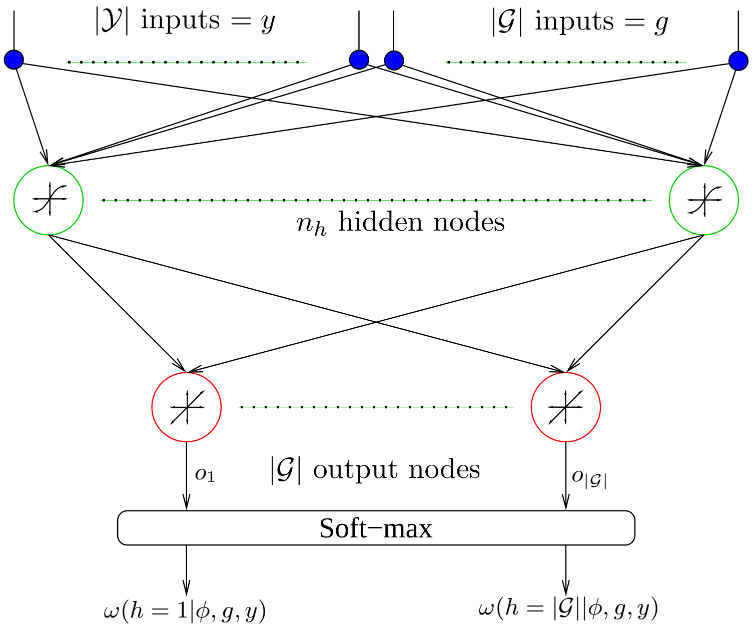
$$\begin{aligned}
\frac{\frac{\partial \omega(h|\phi,g,y)}{\partial \phi_{gy\bar{h}}}}{\omega(h|\phi, g, y)} &= \chi_{\bar{h}}(h) - \omega(\bar{h}|\phi, g, y), \qquad (3.10) \\
\frac{\frac{\partial \mu(u|\theta,h,y)}{\partial \theta_{hy\bar{u}}}}{\mu(u|\theta, h, y)} &= \chi_{\bar{u}}(u) - \mu(\bar{u}|\theta, h, y).
\end{aligned}$$

We have computed the gradient of the *log* of $\omega(h|\phi, g, y)$ and $\mu(u|\theta, h, y)$, which is the quantity required by our algorithms. When lookup tables are infeasible, usually due to large or infinite $|\mathcal{Y}|$, we will use artificial neural networks to parameterise the agent.

3.4.2 Artificial Neural Networks

The lookup-table controller uses $|\mathcal{G}|^2|\mathcal{Y}|$ parameters to encode $\omega(h|\phi, g, y)$ and $|\mathcal{G}||\mathcal{Y}||\mathcal{U}|$ parameters to encode $\mu(u|\theta, h, y)$. If we wish to use many I-states, or there are many (possibly continuous) observations, the lookup table approach falls prey to the curse of dimensionality. For real-world problems we resort to approximations of tables such as artificial neural networks (ANNs).

There are many conceivable ANN architectures that could be used. Our choice is illustrated by Figure 3.1. The I-state is passed in using 1-in-n encoding, such that



$|\mathcal{Y}|$ inputs $= y$
$|\mathcal{G}|$ inputs $= g$
n_h hidden nodes
o_1
$|\mathcal{G}|$ output nodes
$o_{|\mathcal{G}|}$
Soft−max
$\omega(h = 1|\phi, g, y)$
$\omega(h = |\mathcal{G}||\phi, g, y)$

Figure 3.1: The ANN architecture used to represent $\omega(h|\phi, g, y)$ when lookup tables are impractical. There are $|\mathcal{Y}|$ inputs indicating which input was observed, and $|\mathcal{G}|$ inputs encoding the current I-state. If the observations are continuous and multi-dimensional, then we can replace the $|\mathcal{Y}|$ inputs with an input for each element of the observation vector.

I-state input g_t is 1, and all other I-state inputs are 0. The same scheme can be used for the observations if they are finite, otherwise the observation can be passed directly to the ANN.

Step one in computing the ratios $\frac{\nabla\omega(h|\phi,g,y)}{\omega(h|\phi,g,y)}$ and $\frac{\nabla\mu(u|\theta,h,y)}{\mu(u|\theta,h,y)}$ is evaluating $\omega(\cdot|\phi, g, y)$ and $\mu(\cdot|\theta, h, y)$. Concentrating on $\omega(h|\phi, g, y)$, we evaluate the soft-max distribution for each possible future I-state h using the real-valued ANN outputs $\{o_1, \dots, o_{|\mathcal{G}|}\}$

$$\omega(h|\phi, g, y) = \frac{\exp(o_h)}{\sum_{h' \in \mathcal{G}} \exp(o_{h'})}.$$

Next we chose an h from this distribution. Step two is to compute the log gradient for this choice of h by applying the chain rule

$$\begin{aligned}\frac{1}{\omega(h|\phi, g, y)} \frac{\partial\omega(h|\phi, g, y)}{\partial\phi_l} &= \frac{1}{\omega(h|\phi, g, y)} \sum_{h' \in \mathcal{G}} \frac{\partial\omega(h|\phi, g, y)}{\partial o_{h'}} \frac{\partial o_{h'}}{\partial\phi_l} \\ &= \sum_{h' \in \mathcal{G}} \left(\chi_{h'}(h) - \omega(h'|\phi, g, y)\right) \frac{\partial o_{h'}}{\partial\phi_l}.\end{aligned}$$

The first factor in the summation is derived from Equation (3.9) and the second factor is the gradient of the ANN outputs with respect to each ANN weight. The whole expression is implemented similarly to *error back propagation*, which is a standard procedure for training ANNs [Haykin, 1999]. However, instead of back propagating the gradient of an error measure, we back propagate the soft-max gradient for the agent's choice of h. We derive $\frac{\nabla\mu(u|\theta,h,y)}{\mu(u|\theta,h,y)}$ in the same way, evaluating the soft-max distribution for each possible action u by using the real-valued ANN outputs $\{o_1, \ldots, o_{|\mathcal{U}|}\}$.

3.5 Conjugate Gradient Ascent

This section presents an overview of the gradient ascent procedure used for all the experiments reported in this thesis. The methods are not novel so detailed descriptions are deferred to Appendix B.

All the experiments reported in this thesis used the Polak-Ribiére conjugate gradient ascent algorithm [Fine, 1999, §5.5.2] described in Appendix B.1. This algorithm returns a search direction θ^* (which we assume encompasses all parameters) that is orthogonal to any previous search direction.

Unless stated, all experiments used the GSEARCH algorithm [Baxter et al., 2001a] to conduct a line search along the direction θ^* for the best step size γ. Traditional line search methods estimate η for each trial step size, attempting to find two values of γ that bracket the maximum of $\eta(\theta + \gamma\theta^*)$. Instead, GSEARCH computes the sign of the dot product between the search direction and local gradient estimates for each trial step size. When the sign of the dot product changes, it indicates that we stepped past the local maximum we are searching for. Empirically, GSEARCH is more robust in the presence of noise than value-bracketing methods. Reasons for this are discussed in Appendix B.1, along with the details of the GSEARCH algorithm.

Quadratic weight penalties [Kushner and Clark, 1978, §5.2] were used in our experiments to prevent the parameters entering sub-optimal maxima of the soft-max function. Maxima occur whenever the parameters grow large, saturating the soft-max function. See Appendix B.2 for details. The benefits of using penalty terms is demonstrated and discussed as part of the first experiment in Section 4.5.1.

3.6 Summary

Key Points

I The FSC and the world-state induce a global-state Markov chain with transition matrix $P(\phi, \theta) = [p(j, h|\phi, \theta, i, g)]$.

II If $P(\phi, \theta)$ has a unique stationary distribution $\pi(\phi, \theta)$, and if $\omega(h|\phi, g, y)$,

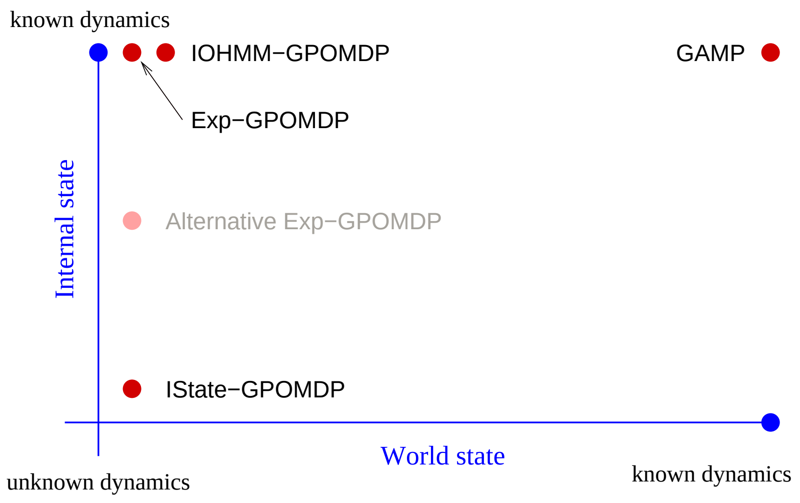


Figure 3.2: The relationship between the algorithms covered in the next three chapters. If a model of the world is available then we can use the algorithms on the right of the world-state axis, otherwise we are restricted to the left hand side. On the left there is a choice of algorithms that make varying degrees of use of the internal-state model.

$\mu(u|\theta, h, y)$, and $r(i)$ are well behaved, then the derivative of the long-term average reward $\nabla\eta$ exists.

III Distributions generated using the soft-max function allow deterministic policies. They are used for most of the experiments reported in this thesis.

IV We use a standard conjugate gradient ascent method. The line search is unusual because it avoids using estimates of η, improving the robustness of the line search.

Sneak Preview

The next three chapters present our novel policy-gradient algorithms that were originally presented in Aberdeen and Baxter [2002]. All can incorporate internal state to allow them to cope with partially observable environments. Figure 3.2 classifies the algorithms by their use of knowledge of the global dynamics. On right of the world-state axis $q(j|i,u)$, $\nu(y|i)$, and $r(i)$, are assumed to be known. On the left the dynamics are completely hidden. Similarly, the algorithms on top of the internal-state axis make use of knowledge of the internal-state transition probabilities $\omega(h|\phi, g, y)$.

The GAMP algorithm is covered in Chapter 4. The IState-GPOMDP algorithm is covered in Chapter 5. The remaining algorithms in the top-left of Figure 3.2 are covered in Chapter 6.

xd

Chapter 4

Model-Based Policy Gradient

Research is what I'm doing when I don't know what I'm doing.

—Wernher von Braun

Without a model of the world there is little choice but to learn through interaction with the world. However, if we are able to at least approximately model the world then gradients can be computed without simulation. For example, manufacturing plants may be reasonably well modeled by hand or models can be estimated using methods from state identification theory [Ogata, 1990]. Given a model we can compute zero-variance gradient approximations quickly and with less bias than Monte-Carlo methods. This chapter introduces one such approximation: the GAMP algorithm. GAMP is feasible for many thousands of states. This is an order of magnitude improvement over model-based value-function algorithms that can handle tens to hundreds of states [Geffner and Bonet, 1998].

We begin with a generic description of how $\nabla\eta$ is computed analytically, then in Section 4.2 we describe GAMP, followed by some experimental results in Sections 4.4 and 4.5.

4.1 Computing $\nabla\eta$ with Internal State

For the purposes of this discussion the model of the POMDP is represented by the global-state transition matrix $P(\phi, \theta)$. The parameter vector ϕ parameterises the FSC model $\omega(h|\phi, g, y)$ and θ parameterises the policy model $\mu(u|\theta, h, y)$. The entries of $P(\phi, \theta)$ are given by Equation (3.1). This matrix has square dimension $|\mathcal{S}||\mathcal{G}|$ and incorporates our knowledge of the world-state transitions given by $q(j|i, u)$, the observation hiding process $\nu(y|i)$, the reward $r(i)$, and the current parameters θ and ϕ.

Recall that the stationary distribution of the global state is $\pi(\phi, \theta)$, a column vector of length $|\mathcal{S}||\mathcal{G}|$. The reward in each global state is assumed known and given by the column vector r. Let e be a length $|\mathcal{S}||\mathcal{G}|$ column vector of all ones. Dropping the explicit dependence on ϕ and θ, $\pi'e$ is a scalar with value 1. The symbol $'$ is used to denote the transpose of a matrix. Also, $e\pi'$ is the outer product of e with π, that is, a rank-1 $|\mathcal{S}||\mathcal{G}| \times |\mathcal{S}||\mathcal{G}|$ matrix with the stationary distribution in each row.

We now derive an exact expression for the gradient of η with respect to the agent's parameters. The rest of this section follows the original derivation by Baxter and Bartlett [2001]. We start by rewriting the scalar long-term average reward (3.7) and its gradient as

$$\begin{aligned} \eta &= \pi' r \\ \nabla\eta &= (\nabla\pi')r. \end{aligned} \tag{4.1}$$

We can derive an expression for $(\nabla\pi')$ by differentiating both sides of the balance equation (3.6)

$$\begin{aligned} \nabla\pi' &= \nabla(\pi' P) \\ &= (\nabla\pi')P + \pi'(\nabla P) \\ \nabla\pi' - (\nabla\pi')P &= \pi'(\nabla P) \\ (\nabla\pi')\,[I-P] &= \pi'(\nabla P), \end{aligned} \tag{4.2}$$

which should be understood as a set of linear equations for each of the $n_\phi + n_\theta$ parameters. For example, for parameter θ_c we have

$$\begin{bmatrix} \frac{\partial\pi(1,1)}{\partial\theta_c} & \cdots & \frac{\partial\pi(|\mathcal{S}|,|\mathcal{G}|)}{\partial\theta_c} \end{bmatrix} \begin{bmatrix} 1-p(1,1|\phi,\theta,1,1) & \cdots & -p(|\mathcal{S}|,|\mathcal{G}||\phi,\theta,1,1) \\ \vdots & \ddots & \vdots \\ -p(1,1|\phi,\theta,|\mathcal{S}|,|\mathcal{G}|) & \cdots & 1-p(|\mathcal{S}|,|\mathcal{G}||\phi,\theta,|\mathcal{G}|,|\mathcal{S}|) \end{bmatrix} =$$

$$\begin{bmatrix} \pi(1,1) & \cdots & \pi(|\mathcal{S}|,|\mathcal{G}|) \end{bmatrix} \begin{bmatrix} \frac{\partial p(1,1|\phi,\theta,1,1)}{\partial\theta_c} & \cdots & \frac{\partial p(|\mathcal{S}|,|\mathcal{G}||\phi,\theta,1,1)}{\partial\theta_c} \\ \vdots & \ddots & \vdots \\ \frac{\partial p(1,1|\phi,\theta,1,1)}{\partial\theta_c} & \cdots & \frac{\partial p(|\mathcal{S}|,|\mathcal{G}||\phi,\theta,|\mathcal{S}|,|\mathcal{G}|)}{\partial\theta_c} \end{bmatrix}.$$

This system is under-constrained because $[I-P]$ is not invertible; which can be shown by re-arranging the balance equation to reveal a leading left eigenvector with zero eigenvalue (all 0 vectors and matrices are represented by $[0]$)

$$\begin{aligned} \pi' &= \pi' P \\ \pi'[I-P] &= [0]. \end{aligned} \tag{4.3}$$

We avoid this problem by conditioning $[I-P]$ as follows. Recall that $e\pi'$ is the

$|\mathcal{S}||\mathcal{G}| \times |\mathcal{S}||\mathcal{G}|$ matrix with the stationary distribution π' in each row. Since

$$\begin{aligned}(\nabla\pi')e &= \sum_{i,g} \nabla\pi(i,g)\\ &= \nabla \sum_{i,g} \pi(i,g)\\ &= \nabla 1\\ &= 0,\end{aligned}$$

we obtain $(\nabla\pi')e\pi' = [0]$. Thus, adding $e\pi'$ to $I - P$ adds 0 to $(\nabla\pi')\,[I-P]$ and we can rewrite (4.2) as

$$(\nabla\pi')\left[I - P + e\pi'\right] = \pi'(\nabla P).$$

To show that $[I - P + e\pi']$ is invertible we call upon a classic matrix theorem:

Theorem 1 (Theorem 1, §4.5, Kincaid and Cheney [1991]). *Let A be an $n \times n$ matrix with elements a_{ij}. Let $\|A\|_p$ be the subordinate matrix norm induced by the vector p-norm. For example*

$$\|A\|_\infty := \max_i \sum_{j=1}^{n} |a_{ij}|,$$

then for any p, if $\lim_{n\to\infty} \|A^n\|_p = 0$*, we have*

$$[I - A]^{-1} = \sum_{n=0}^{\infty} A^n. \tag{4.4}$$

We now demonstrate that $A = (P - e\pi')^n$ converges to $[0]$ as $n \to \infty$, hence that $[I - (P - e\pi')]$ is invertible. The first step is showing $(P - e\pi')^n = P^n - e\pi'$. This is trivially true for $n = 1$, now we assume it is true for some n and demonstrate it is true for $n + 1$

$$\begin{aligned}(P - e\pi')^{n+1} &= (P^n - e\pi')(P - e\pi') \quad \text{, from assumption}\\ &= P^{n+1} - e\pi'P - P^n e\pi' + e\pi'e\pi'\\ &= P^{n+1} - e\pi' - P^n e\pi' + e1\pi' \quad \text{, from } \pi'P = \pi' \text{ and } \pi'e = 1\\ &= P^{n+1} - P^n e\pi'\\ &= P^{n+1} - e\pi' \quad \text{, from } Pe = e,\end{aligned} \tag{4.5}$$

and by induction it is true for all n. As $n \to \infty$ we have $P^n \to e\pi'$, so

$$\lim_{n\to\infty} A^n = \lim_{n\to\infty} P^n - e\pi' = e\pi' - e\pi' = [0].$$

Thus $[I - (P - e\pi')]$ is invertible and we can write

$$(\nabla\pi') = \pi'(\nabla P)\left[I - P + e\pi'\right]^{-1}.$$

So, applying Equation (4.1),

$$\nabla\eta = \pi'(\nabla P)\left[I - P + e\pi'\right]^{-1} r. \tag{4.6}$$

4.2 The GAMP Algorithm

Computing $[I - P + e\pi']^{-1}$ exactly is $O(|\mathcal{S}|^3|\mathcal{G}|^3)$ hence intractable for more than a few 100's of states. However, we can closely approximate the inversion using a series matrix expansion that, when taking advantage of sparse data structures, becomes feasible for many thousands of states. Similarly, we approximate π by iteration instead of computing the leading left eigenvector of P. Using these two approximations is the essence of the Gradient Approximation for Modeled POMDPs (GAMP) method outlined by Algorithm 1.

Using (4.4), plus the fact that $(P - e\pi')^n = P^n - e\pi'$, we can rewrite Equation (4.6) as

$$\begin{aligned}
\nabla\eta &= \lim_{N\to\infty} \pi'(\nabla P)\left[\sum_{n=0}^{N}\left(P^n - e\pi'\right)\right] r && (4.7)\\
&= \lim_{N\to\infty} \pi'(\nabla P)\left[\left(\sum_{n=0}^{N} P^n\right) - \left(\sum_{n=0}^{N} e\pi'\right)\right] r\\
&= \lim_{N\to\infty} \pi'\left[(\nabla P)\left(\sum_{n=0}^{N} P^n\right) - \left(\sum_{n=0}^{N}(\nabla P)e\pi'\right)\right] r\\
&= \lim_{N\to\infty} \pi'(\nabla P)\sum_{n=0}^{N} P^n r, && (4.8)
\end{aligned}$$

where the last line follows from $(\nabla P)e = [0]$ for the same reason as $(\nabla\pi')e = 0$. Let x_N be the summation up to the N'th term

$$x_N = \sum_{n=0}^{N} P^n r. \tag{4.9}$$

Now we simply define the GAMP gradient approximation as Equation (4.8) with the truncated summation x_N

$$\widehat{\nabla_N \eta} := \pi'(\nabla P)x_N.$$

We evaluate x_N by iterating

$$\begin{aligned} v_0 &= r, \quad v_{n+1} = Pv_n \\ x_0 &= [0], \quad x_{n+1} = x_n + v_n. \end{aligned}$$

Because this is a series of matrix-vector multiplications, and matrix additions, we end up with an algorithm that has worst case complexity $O(|\mathcal{S}|^2|\mathcal{G}|^2 N)$. This is a form of Richardson iteration, a simple iterative method for solving systems of linear equations [Kincaid and Cheney, 1991, §4.6].

The matrix P is usually sparse since only a small number of states $j \in \mathcal{S}$ have non-zero probabilities of being reached from some state i. For example, in the robot navigation domain of Cassandra [1998] the move forward action leads to one of at most 3 next states, regardless of the size of the world. Using sparse matrix data structures and sparse multiplication algorithms, the practical complexity is $O(c|\mathcal{S}||\mathcal{G}|N)$ where $c \ll |\mathcal{S}||\mathcal{G}|$ and depends on the degree of sparsity exhibited by P.

Evaluating π involves computing the leading left eigenvector of P, which is expensive if done exactly. We use the power method [Anton and Rorres, 1991, §9.5] which comprises of iterating $\pi'_{n+1} = \pi'_n P$. We stop the iteration when the error $\|\pi'_{n+1} - \pi'_n\|_\infty$ falls below some threshold ϵ_π. Sparse matrix multiplication again reduces the complexity to $O(c|\mathcal{S}||\mathcal{G}|n)$ where $c \ll |\mathcal{S}||\mathcal{G}|$.

A surprisingly expensive operation is evaluating Equations (3.4) and (3.5) for each element of each matrix $\partial P/\partial\phi_l$ and each element of each matrix $\partial P/\partial\theta_c$. In the worst case this has complexity $O(|\mathcal{S}|^2|\mathcal{G}|^2(n_\phi + n_\theta)|\mathcal{Y}||\mathcal{U}|)$. Since $q(j|i,u)$ and $\nu(y|i)$ are often 0, practical complexities are $O(c|\mathcal{S}||\mathcal{G}|(n_\phi + n_\theta)|\mathcal{U}|)$ where $c \ll |\mathcal{S}||\mathcal{G}||\mathcal{Y}|$. Simple tricks such as pre-computing all values of $\nabla\omega(h|\phi, g, y)$ and $\nabla\mu(u|\theta, h, y)$, combining the operations of computing ∇P with multiplication by π' and x_N, and using sparsity, allows systems of $|\mathcal{G}||\mathcal{S}| > 20,000$ to be feasibly tackled on modern desktop computers.

With reference to Algorithm 1, lines 2–7 compute P; 8–13 compute x_N using Richardson iteration; 14–18 estimate π' using the power method and 19–29 compute $\pi'(\nabla P)x_N$. Combining the computation of ∇P with the final step of multiplying $\pi'(\nabla P)x_N$ avoids explicitly storing $n_\phi + n_\theta$ matrices that each have $|\mathcal{S}|^2|\mathcal{G}|^2$ elements. Practical implementations require sparse data representations and matrix multiplications. The loops for computing P and ∇P in Algorithm 1 are shown in a simplified form for clarity. They should be constructed to take maximum advantage of the factors in lines 5, 11, 23, and 26 that are often 0.[1]

4.3 Asymptotic Convergence of GAMP

Because P is an ergodic matrix, P^N converges exponentially quickly to $e\pi'$. The exact rate is governed by the mixing time τ of the POMDP, which we define similarly to Barlett and Baxter [2000].

Definition 3. *The total variation distance between two discrete probability distributions P, Q on a set $\mathcal{S}$ is*

$$d(p,q) = \sum_{j\in\mathcal{S}} |p(j) - q(j)|.$$

Definition 4. *We say that a stochastic process with unique stationary distribution $\pi(\phi,\theta)$, is exponentially mixing with time constant τ (τ-mixing for short) if $\forall i$*

$$d(P_i^N, \pi) \le \exp(-\lfloor \frac{N}{\tau} \rfloor), \tag{4.10}$$

where P_i^N is the i'th row of P^N. The mixing time of the POMDP is defined to be the smallest τ such that P is τ-mixing.

Intuitively, τ can be thought of as a measure of how long it takes to obtain a "representative" sample from the Markov chain. The mixing time depends on the stationary distribution, which depends on the current agent parameters. Thus, as the agent's policy evolves, the mixing time changes. The following theorem bounds the error in the GAMP gradient estimates as a function of τ. Recall from Section 4.1 that R bounds rewards, Q bounds $\nabla\omega(h|\phi,g,y)$ and U bounds $\nabla\mu(u|\theta,h,y)$.

Theorem 2.

$$\|\nabla^{\phi}\eta - \widehat{\nabla_N^{\phi}\eta}\|_\infty \le QR|\mathcal{G}|\tau\frac{\exp(-\lfloor\frac{N}{\tau}\rfloor)}{1-\exp(-1)}$$
$$\|\nabla^{\theta}\eta - \widehat{\nabla_N^{\theta}\eta}\|_\infty \le UR|\mathcal{U}|\tau\frac{\exp(-\lfloor\frac{N}{\tau}\rfloor)}{1-\exp(-1)}$$

This theorem is proved in Appendix A.1.1. The difficulty of calculating τ for an arbitrary POMDP makes it hard to use this theorem to establish N in advance. In

[1]Specifically, thought must be given to the loop ordering. The most complex loop is lines 19–29. From outer-most to inner-most we loop over $i, y, g, h, \{\theta_c, \phi_l\}, u, j$. The inner loops are only entered if none of the factors in lines 23 or 26 have already evaluated to 0. For example, the two outermost loops over i and y allow us to evaluate $\nu(y|i)$, which, if it is 0, means we increment y without entering the deeper loops. Sparse matrix representations of $q(j|i,u)$ make the inner-most loop efficient.

Algorithm 1 GAMP

```
1: Given:
     • State transition probabilities $q(j|i,u) \quad \forall j,i \in \mathcal{S}, u \in \mathcal{U}$.
     • Observation probabilities $\nu(y|i) \quad \forall i \in \mathcal{S}, y \in \mathcal{Y}$.
     • Rewards $r(i,g) \quad \forall i \in \mathcal{S}, g \in \mathcal{G}$.
     • Policy $\mu(u|\theta,h,y)$ parameterised by $\theta \in \mathbb{R}^{n_\theta}$.
     • FSC $\omega(h|\phi,g,y)$ parameterised by $\phi \in \mathbb{R}^{n_\phi}$.
     • Iteration termination thresholds $\epsilon_x, \epsilon_\pi$.
2: for each $(i,g),(j,h)$ do
3:   $p_{(i,g)(j,h)} = 0$
4:   for each $\{(y,u)|\nu(y|i)\omega(h|\phi,g,y)\mu(u|\theta,h,y)q(j|i,u) \neq 0\}$ do
5:     $p_{(i,g)(j,h)} = p_{(i,g)(j,h)} + \nu(y|i)\omega(h|\phi,g,y)\mu(u|\theta,h,y)q(j|i,u)$
6:   end for
7: end for
8: $v = r, x = r, \bar{x} = 0$
9: while $\max_{(i,g)} |x(i,g) - \bar{x}(i,g)| > \epsilon_x$ do
10:   $\bar{x} = x$
11:   $v = Pv$
12:   $x = x + v$
13: end while
14: $\bar{\pi} = \frac{1}{|\mathcal{G}||\mathcal{S}|}, \pi' = \bar{\pi}P$
15: while $\max_{(i,g)} |\pi(i,g) - \bar{\pi}(i,g)| > \epsilon_\pi$ do
16:   $\bar{\pi} = \pi'$
17:   $\pi' = \bar{\pi}P$
18: end while
19: $\Delta = [\Delta^\theta, \Delta^\phi] = [0]$
20: for each $\{(i,g),(j,h)|x(j,h) \neq 0\})$ do
21:   for each $\{(y,u)|\nu(y|i)q(j|i,u) \neq 0\}$ do
22:     for each $\{\theta_c \in \theta|\omega(h|\phi,g,y)\frac{\partial\mu(u|\theta,h,y)}{\partial\theta_c} \neq 0\}$ do
23:       $\Delta_c^\theta = \Delta_c^\theta + \pi(i,g)\nu(y|i)\omega(h|\phi,g,y)\frac{\partial\mu(u|\theta,h,y)}{\partial\theta_c}q(j|i,u)x(j,h)$
24:     end for
25:     for each $\{\phi_l \in \phi|\frac{\partial\omega(h|\phi,g,y)}{\partial\phi_l}\mu(u|\theta,h,y) \neq 0\}$ do
26:       $\Delta_l^\phi = \Delta_l^\phi + \pi(i,g)\nu(y|i)\frac{\partial\omega(h|\phi,g,y)}{\partial\phi_l}\mu(u|\theta,h,y)q(j|i,u)x(j,h)$
27:     end for
28:   end for
29: end for
30: $\widehat{\nabla_N\eta} = \Delta$
```

practice we check for convergence of x by stopping when $\|x_{N+1} - x_N\|_\infty \leq \epsilon$. The following theorem shows that $\|x_{N+1} - x_N\|_\infty$ is decreasing; a necessary property if it is to be used as stopping criterion.

Theorem 3.

$$\|x_{N+1} - x_N\|_\infty \leq \|x_N - x_{N-1}\|_\infty. \tag{4.11}$$

The proof is straightforward and located in Appendix A.1.2.

4.4 GAMP in Practice

Figure 4.4 demonstrates empirically how quickly the GAMP algorithm converges on an internal-state problem with 1045 global states. We computed the exact matrix inversion $[I - P + e\pi']^{-1}$ for the Pentagon problem with $|\mathcal{S}| = 209$ and $|\mathcal{G}| = 5$. The details of this scenario are deferred until Section 8.3 since they are not important for understanding this experiment. The agents were parameterised with tables of real numbers as described in Section 3.4.1. The initial parameters were set such that $\theta_c = \phi_l = 0, \ \forall c, l$.

The Pentagon problem allows all observations from all states by adding observation noise, removing the gain we would normally achieve from sparse observations. Noise is also added to the world-state transitions. The added noise and internal state make this a good challenge for GAMP but, despite noise, the global-state transition probabilities are still sparse with only 25,875 of 1,092,025 elements of P having non-zero probabilities.

This experiment was run on an unloaded Pentium II @ 433 MHz. When computing the exact gradient, finding π' requires 315 s (wall clock time),[2] the matrix inversion requires 10.5 s and ∇P requires 36 s. When computing the approximate gradient with $N = 500$, the Richarsdon Iteration inversion requires 1.41 s. A larger saving comes from approximating π'. For this experiment we used a π iteration stopping threshold of $\epsilon_\pi = 0.0001$. This required 1319 iterations, taking 3.50 s instead of 315 s. The angular error in the gradient at $N = 500$ is 0.420° taking 11.3% of the time the true gradient requires. The speedup becomes greater as $|\mathcal{S}||\mathcal{G}|$ grows. If π' is computed exactly, but Richardson Iteration is still used, the error is reduced by 0.016°, demonstrating that in this case approximating π accounts for only a small portion of the error.

[2]We used the `dgeev` routine of LAPACK to compute *all* eigenvectors exactly. We could determine π faster with an algorithm that exactly computes only the leading left eigenvector.

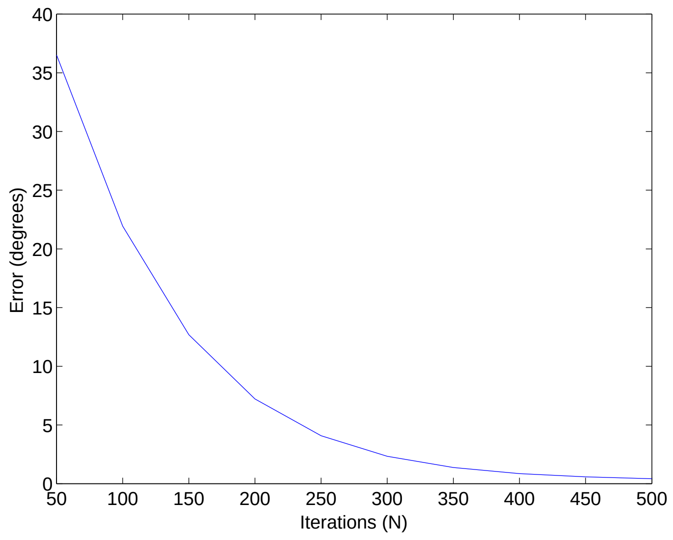


Figure 4.1: Angular error between the GAMP estimate after N iterations and the exact gradient. This plot is based on the Pentagon POMDP with $|\mathcal{S}| = 209$ and $|\mathcal{G}| = 5$ making P a 1045×1045 element matrix.

4.5 A Large Multi-Agent Problem

Model based methods for POMDPs have been restricted to at most a few hundred states with 10's of observations and actions [Geffner and Bonet, 1998]. This section demonstrates that GAMP can learn the optimal policy for a noisy multi-agent POMDP with 21,632 states, 1024 observations and 16 actions.

The scenario is shown in Figure 4.2: a factory floor with 13 grid locations to which 2 robots have access. The robots are identical except that one is given priority in situations where both robots want to move into the same space. They can turn left or right, move 1 position ahead, or wait where they are. One agent learns to move unfinished parts from the left shaded area to the middle area, where the part is processed instantly, ready for the second agent to move the processed part from the middle to the right shaded area. The middle processing machine can only handle 1 part at a time, so if the first agent drops off a part at the middle before the second agent has picked up the last part dropped at the middle, the new part is discarded.

The large state space arises from the combined state of the two independent agents plus the global state. Each agent can be loaded or unloaded in 13 states with 4 orientations, giving each agent $2 \times 13 \times 4 = 104$ states. The global state indicates if a part is waiting at the middle processing machine and the state of the 2 agents, giving $2 \times 104^2 = 21,632$ states.

A reward of 1 is received for dropping off a processed part at the right. The agents

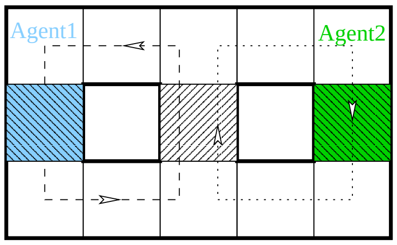


Figure 4.2: Plan of the factory floor for the multi-agent problem. The dashed arrows shows one of the routes traced by the final agents.

only need to exit the loading or drop off locations to pick up or drop loads. To receive the maximum reward the agents must cooperate without explicit communication, the actions of the first agent allowing the second agent to receive rewards.

The observations for each agent consist of 4 bits describing whether their path is blocked in each of the 4 neighbouring positions, and a 5th bit describing if the agent is in the uppermost corridor (which is necessary to break the symmetry of the map). The combined observations are 10 bits, or $|\mathcal{Y}| = 1024$.[3] The actions for each agent are {`move forward, turn left, turn right, wait`}, resulting in a total of $|\mathcal{U}| = 16$ actions.

Uncertainty is added with a 10% chance of the agents' action failing, resulting in no movement, and a 10% chance of the agents' sensors completely failing, receiving a "no walls" observation. This problem was designed to be solved by a reactive policy. Section 8 demonstrates GAMP on problems that require memory to solve.

4.5.1 Experimental Protocol

These experiments were run on an unloaded AMD Athlon @ 1.3 GHz. GAMP required less than 47 Mbytes of RAM to run this problem. Compare this to just storing every element of ∇P explicitly, which would require 893 Giga bytes of ram.

The agents were parameterised with tables of real numbers as described in Section 3.4.1. There are $|\mathcal{Y}| \times |\mathcal{U}| = 128$ parameters per agent. We set $\theta_c = 0 \; \forall c$. There are no ϕ parameters since the scenario can be solved without I-states. A quadratic penalty of $\wp = 0.0001$ was used to stop the parameters settling in a local maximum too early (see Appendix B.2). The quadratic penalties for all the experiments in this thesis were chosen by trial and error. We determined penalties that prevented the weights growing past approximately 0.5 before the penalty is automatically reduced for the first time. Penalty reduction occurs after three line search iterations without improvement of the average reward. We chose $\epsilon_x = \epsilon_\pi = 0.0001$, which was the largest value (for

[3]Not all 1024 observations can occur, contributing the sparseness of the system.

Table 4.1: Results for multi-agent factory setting POMDP. The values for η are multiplied by 10^2.

Algorithm	*mean* η	*max.* η	*var.*	*secs to* $\eta = 5.0$
GAMP	6.51	6.51	0	1035
Hand	6.51			

the fastest approximation) tested that allowed the agent to consistently converge to an agent with equivalent performance to the best hand coded policy.

Exact algorithms based on factored belief states could work well for this scenario since it decomposes into a *state variables* [Boutilier and Poole, 1996, Hansen and Feng, 2000, Poupart and Boutilier, 2001], however we do not assume that the state-variable model is known. We shall discuss some possibilities for factored versions of GAMP at the end of this chapter.

4.5.2 Results

The agents learnt to move in opposing circles around the factory (shown by the dashed lines in Figure 4.2). This policy reduces the chances of collision. They also learnt to wait when their sensors fail, using the wait action to gather information. Table 4.1 shows a comparison between GAMP with no memory and the best policy we designed by hand.[4]

Without applying quadratic penalties training terminated in substantially sub-optimal local maxima. This is because the early gradient estimates tended to point in sub-optimal directions, dominated by concepts that are easy to learn, such as "don't run into walls," or "moving forward is good." These gradients drive parameters to very large values. The soft-max function enters a local maximum when the parameters diverge to $\pm\infty$, so the agent quickly becomes stuck having learnt only the most simple concepts. The quadratic penalty keeps parameters near 0. This forces the $\omega(\cdot|\phi, g, y)$ and $\mu(\cdot|\theta, h, y)$ distributions to stay close to uniform, which encourages exploration. The most common local maxima occurred when the agents learnt early in training that `forward` is a generally useful action, even when the sensors fail and the agent should wait for more information. Because the move forward concept was learnt so strongly the soft-max derivative for the relevant parameters was close to 0.

We attempted to run the exact Incremental Pruning algorithm [Zhang and Liu, 1996] on the Factory problem. The code aborted during the first iteration of dynamic

[4]An mpeg visualisation of trained agents is available from the author, or from `http://discus.anu.edu.au/~daa/files/factory.mpg`.

programming after consuming all 256 MB of memory.[5] Storing just one double precision belief state of length 21,632 requires 169 Kb and exact value-function-based algorithms quickly generate many thousands of vectors for large problems.

4.6 Discussion

Even though GAMP enumerates the state space it does not suffer from the curse of dimensionality as severely as exact methods because of three features: local optimisation, approximations to the gradient, and the ability to take full advantage of sparsity in the POMDP model.

However, if the state space grows very large then simulation will out-perform GAMP unless a form of state space pruning is used. This arises from a nice feature of Monte-Carlo algorithms: their ability to focus search on the relevant parts of state space [Kaelbling et al., 1996]. Except in the early stages of training, Monte-Carlo methods encounter only those states that are entered by a reasonable agent, effectively learning in a pruned state space. GAMP always considers the entire state space, even those states that are unreachable given a start state.

For this reason it is important to ensure Assumption 1 from Section 3.2 is not violated when applying GAMP. For example, when designing the Factory problem, care was taken to ensure that impossible situations, such as two robots occupying the same location, have transitions to legal situations even though Monte-Carlo algorithms would not enter those states. This discussion assumes that the regions of state space ignored by Monte-Carlo algorithms are truly irrelevant to performance, an assumption that is often violated. Quadratic penalties are also useful to ensure that Monte-Carlo methods do not ignore regions of state space without sufficient exploration.

To scale to hundreds-of-thousands of states more advanced iteration methods are worthy of investigation. For example, the GMRES algorithm is a Krylov subspace method for computing x_N. It is similar to Richardson iteration but computes estimates of x_{N+1} based on multiple previous estimates $\{x_0, \dots, x_N\}$ [Greenbaum, 1997, §2][Ipsen and Meyer, 1998]. For computing π, the *Lanczos method* [Greenbaum, 1997, §2.5], which is related to Krylov subspace methods, is available.

Finally, we emphasise that the advantage of GAMP lies in its ability to compute gradients of η without interacting with the world. Interactions are slow and can be expensive and dangerous. Fewer samples may be needed to estimate the model parameters, prior to running GAMP, than to run model-free algorithms. For example, an agent that learns to drive a car using Monte-Carlo methods will require thousands of driving hours and could crash a few times along the way. However, sophisticated driv-

[5]We used Anthony Cassandra's `pomdp-solve` V4.0 code.

ing models exist that take into account car dynamics, driving conditions, and traffic; for example, state-of-the-art computer games. A better approach is to let the agent learn off-line using the approximate model, and then refine its policy on the road.

4.7 Related Work

Monte-Carlo policy-gradient algorithms have been well studied, but little work has examined policy-gradient approaches when given a model. The Linear-Q [Littman et al., 1995] and SPOVA [Parr and Russell, 1995] algorithms use gradient ascent to perform parameter updates for function approximators that learn *values* of belief states. Apart from being value-based, this work differs from GAMP because simulation is used, incurring the associated variance penalty but working for very large state spaces.

Meuleau et al. [1999a] used a similar approach to ours, computing the gradient of the *discounted* reward J_β. They avoid computing π because of discounting, but this necessitates an iterative matrix inverse for each parameter. During each gradient calculation GAMP performs two iterative solves compared to the $n_\phi + n_\theta$ matrix inverses performed to compute ∇J_β. The paper of Meuleau noted that the VAPS Monte-Carlo approach out-performed the exact gradient as the discount factor approached 1, even for the trivial Load/Unload scenario, limiting its usefulness to larger scenarios that require high discount factors. As we demonstrate in Section 8, GAMP outperforms our Monte-Carlo approaches on a variety of infinite-horizon scenarios, without using discount factors.

4.8 Summary

Key Points

- I GAMP uses a discrete model of the world to approximate $\nabla\eta$.
- II GAMP does not need to interact with the world during learning.
- III GAMP can be used to train agents for worlds with tens-of-thousands of states on a modern desktop computer.
- IV GAMP can be applied to discretised versions of continuous state, observation and action spaces.

Future Work

Computing ∇P is the current bottleneck. Using factored representations of the POMDP (Section 2.5.3) will speed up computation of ∇P and all other phases due to compact representations of the matrices in terms of state variables. This may introduce further approximation error due to imperfect factorisation. One method for constructing matrices from state variables is to use *algebraic decision diagrams* (ADDs). Methods for performing matrix addition and multiplication using ADDs are given by Bahar et al. [1993], and ADDs have been previously applied to MDP value methods by Hoey et al. [1999].

GMRES for computing x_N and the Lanczos method of computing π will be beneficial for scaling to larger POMDPs than the factory problem.

If the model of the world is only approximate then GAMP can be used during a preliminary training phase. Once a “rough draft” agent has been produced using GAMP and the approximate model, it could be refined in the real world by using the Monte-Carlo algorithms described in following chapters.

Chapter 5

Model-Free Policy Gradient

Those who can, do. Those who can't, simulate.
—Anon.

Without knowledge of the POMDP transition and observation model we resort to Monte-Carlo methods for estimating $\nabla\eta(\phi, \theta)$. These methods experience trajectories through the POMDP, gathering noisy information about the true gradient at each step. Much work has been done in this area for MDPs, including the value-based $\mathsf{TD}(\lambda)$ [Sutton and Barto, 1998] and $\mathsf{SARSA}(\lambda)$ (see Appendix C.1.1) families of algorithms.

As discussed in Section 2.2, agents require internal state to act optimally in partially observable environments. The key idea of this chapter is to augment the world state with a set of fully observable internal states that the agent can manipulate to use as memory. Through Monte-Carlo trials, the agent learns to use its finite state controller (FSC) to update the internal state in a useful way. Initially the agent's internal state transitions are random and do not help the agent. However, the internal state transition parameters ϕ adjust and learn to record parts of the agent's history which are relevant to choosing better actions. Once the internal state becomes relevant to the agent, the action selection parameters θ adjust to take advantage of the internal state, hence increasing the long-term reward.

Peshkin et al. [1999] and Peshkin [2002] used a similar approach in extending Williams' REINFORCE to policies with memory. We now extend this to the infinite-horizon setting by modifying the memory-less GPOMDP algorithm of Baxter and Bartlett [2001]. We shall derive our memory enabled version from scratch so an understanding of GPOMDP is not necessary. However, a brief introduction to GPOMDP can be found in Section 2.6.3.

Experiments are deferred until Chapter 8, where we compare all our algorithms.

5.1 Gathering Experience from the World

The task is now to estimate the gradient of the long-term average reward by interacting with the world. Model-free algorithms are sometimes referred to as *simulation* methods because learning can be implemented by letting the agent interact with a computer simulation of the world. This terminology is slightly misleading because model-free methods usually make no distinction between learning from a simulation of the world, and learning from the real world. For this reason we prefer the term *Monte-Carlo* methods for model-free algorithms that can learn using simulators *or* the real world. In both cases we sample trajectories through the state space, but simulators do this using a random number generator instead of nature's randomness. The distinction is important because Section 9.4 discusses an existing variance reduction method that manipulates the random number generator directly, requiring *simulation* of state transitions. The ability to simulate transitions implies a known model (although the agent does not have access to it).

Interaction begins by placing the agent in a random global state (i_0, g_0). The assumption of a unique stationary distribution means the effect of the initial state on the gradient estimate will become negligible with time. Assuming the agent is in global state (i_t, g_t) at time t, one step of interaction progresses as follows:

1. the agent observes y_t sampled from the distribution $\nu(\cdot|i_t)$;
2. the agent chooses its next I-state g_{t+1} from the distribution $\omega(\cdot|\phi, g_t, y_t)$;
3. the agent chooses an action u_t from the distribution $\mu(\cdot|\theta, g_{t+1}, y_t)$;
4. the next world state is sampled from the distribution $q(\cdot|i_t, u_t)$.
5. the reward $r_{t+1} = r(i_{t+1})$ is sent to the agent;
6. in training mode the agent updates its gradient estimate.

This chapter focuses on the last step.

5.2 The IState-GPOMDP Algorithm

Algorithm 2 *estimates* $\Delta_T := [\Delta_T^\phi, \Delta_T^\theta]$, which in turn *approximates* the gradient $\nabla\eta = [\nabla^\phi\eta, \nabla^\theta\eta]$. We describe the estimate in a bottom-up fashion, describing the algorithm and then providing the theoretical guarantees that the algorithm converges to an estimate of $\nabla\eta$.

At each step an observation y_t is received and the distribution $\omega(\cdot|\phi, g_t, y_t)$ is sampled to choose g_{t+1}. The gradient of $\log\omega(g_{t+1}|\phi, g_t, y_t)$ is added into an eligibility trace z_t^ϕ, which is discounted by $\beta \in [0, 1)$ at each step. The eligibility trace performs the temporal credit assignment, giving more credit to recent I-state choices. At time t

the eligibility trace is equal to the n_ϕ element column vector

$$z_t^\phi = \begin{bmatrix} \frac{\partial \log \omega(g_{t+1}|\phi, g_t, y_t)}{\partial \phi_1} + \beta \frac{\partial \log \omega(g_t|\phi, g_{t-1}, y_{t-1})}{\partial \phi_1} + \beta^2 \frac{\partial \log \omega(g_{t-1}|\phi, g_{t-2}, y_{t-2})}{\partial \phi_1} + \dots \\ \vdots \\ \frac{\partial \log \omega(g_{t+1}|\phi, g_t, y_t)}{\partial \phi_{n_\phi}} + \beta \frac{\partial \log \omega(g_t|\phi, g_{t-1}, y_{t-1})}{\partial \phi_{n_\phi}} + \beta^2 \frac{\partial \log \omega(g_{t-1}|\phi, g_{t-2}, y_{t-2})}{\partial \phi_{n_\phi}} + \dots \end{bmatrix}.$$

The same process is followed with the new I-state to choose an action u_t, adding the gradient of $\log \mu(u_t|\theta, g_{t+1}, y_t)$ to the eligibility trace z_t^θ. At each step the immediate reward after action u_t is multiplied by the current traces and averaged to form the gradient estimate. The discount factor β reflects the assumption that rewards are exponentially more likely to be generated by recent actions. It can be viewed as imposing an artificial horizon on the POMDP. Algorithms such as Williams' REINFORCE avoid discount factors by assuming finite-horizon POMDPs, bounding the period in which actions can affect rewards.

The following theorem establishes that IState-GPOMDP estimates an approximation of $\nabla\eta$. The exact expression for $\nabla\eta$ is given by Equation (4.6). The IState-GPOMDP estimate converges to the approximate gradient as the number of estimation steps goes to ∞. Recall that J_β is the average discounted reward defined by Equation (2.2).

Theorem 4. *Let $\Delta_T := \left[\Delta_T^\theta, \Delta_T^\phi\right]$ be the estimate produced by IState-GPOMDP after T iterations. Then under Assumptions 1–4 from Section 3.2,*

$$\lim_{T\to\infty} \Delta_T = \pi'(\nabla P)J_\beta \text{ with probability 1.}$$

This is proved in Appendix A.2.2. The next theorem establishes that as the discount factor $\beta \to 1$, the approximation is exact.

Theorem 5. *For $P(\theta, \phi)$*

$$\lim_{\beta\to 1} \pi'(\nabla P)J_\beta = \nabla\eta.$$

The proof does not depend on how P is parameterised so it is unchanged from the original memory-less proof [Baxter and Bartlett, 2001] which is reproduced in Appendix A.2.1. Theorem 4 is a generalisation from the memory-less GPOMDP algorithm. GPOMDP is retrieved from IState-GPOMDP by setting $|\mathcal{G}| = 1$.

We now have $\Delta_T \xrightarrow{T\to\infty} \pi'(\nabla P)J_\beta \xrightarrow{\beta\to 1} \nabla\eta$, however the variance of Δ_T scales with $1/\left[T(1-\beta)\right]$. This reflects the increasing difficulty of the credit assignment problem as the horizon lengthens and β increases.

Fortunately, it was proven for the memory-less setting [Baxter and Bartlett, 2001] that $\pi'(\nabla P)J_\beta$ is guaranteed to be a good approximation to $\nabla\eta$ if $1/(1-\beta)$ exceeds τ, modulo constants and log factors. Recall that τ is the mixing time of the transition matrix P defined by Equation (4.10). The same guarantee holds for IState-GPOMDP, but we use the mixing time of the global-state, that is, the mixing time of the cross product of the world-state space with the internal-state space. To see that the guarantee holds, observe that an agent with FSC memory can be transformed into an equivalent memory-less agent. The transformation involves moving the FSC from the agent to the world, allowing the agent to update the FSC state with *external memory*[1] setting actions. We implicitly perform this transformation when we compute the global-state transition matrix $P(\phi, \theta)$. This transformation argument can also be used as a simple alternative proof for Theorem 4 given the theorem is true for the non-memory setting.

5.3 Complexity of IState-GPOMDP

The training complexity of policy-gradient Monte-Carlo algorithms can be factored into two components: the complexity of each step of Algorithm 2, and the number of steps required to gather a good estimate.

5.3.1 Per Step Complexity

The complexity of each step is dominated by the complexity of lines 5–8 in Algorithm 2. The distributions $\omega(\cdot|\phi, g_t, y_t)$ and $\mu(\cdot|\theta, g_{t+1}, y_t)$ must be evaluated and sampled, then the log gradients computed for the samples. The exact complexity depends on the choice of parameterisation. For example, using the lookup table scheme described in Section 3.4.1 the complexity is $O(|\mathcal{G}| + |\mathcal{U}|)$, representing the complexity of evaluating two soft-max functions and their derivatives, one with $|\mathcal{G}|$ parameters and the other with $|\mathcal{U}|$ parameters. The artificial neural network scheme of Section 3.4.2 has complexity $O(|\mathcal{G}| + |\mathcal{U}| + n_\phi + n_\theta)$.

Another example is distributions modeled by a Gaussian, requiring only four parameters — the mean and variance for ω and μ — for each combination of (g, y). In this case the step complexity is the cost of sampling a single variable Gaussian and computing its gradient at the sample point. The gradient is with respect to the mean and variance of the sampled Gaussian.

IState-GPOMDP is memory efficient, requiring a minimum of only $2n_\phi + 2n_\theta$ units of memory: a copy of the current parameters, and the eligibility traces z_t^ϕ and z_t^θ.

[1]The external memory model was used by Peshkin et al. [1999]. It has some appeal because there is anecdotal evidence that people use "auxiliary" memory in a similar way. For example, writing notes to themselves, or leaving medicine bottles in a different place after the medicine has been taken [Examiner, 2003]. We prefer the internal memory model because people have a clear concept of their own memory which is independent of the external world. The internal memory model also provides a clear distinction between the process of updating and using memory, and performing actions which alter the world state. However, the two models are mathematically equivalent.

Compare this with exact algorithms that are PSPACE-complete (Section 2.5.1.4).

5.3.2 Mixing Time

Using $|\mathcal{G}|$ I-states results in a global-state space that is $|\mathcal{G}|$ times larger than the world-state space. This can cause an increase in the mixing time τ. Because $1/(1-\beta)$ must exceed $O(\tau)$ to provide good estimates, and because the variance scales with $1/\left(T(1-\beta)\right)$, we come to the natural conclusion that increasing $|\mathcal{G}|$ can potentially increase the variance of the gradient estimates. We also come to the conclusion that increasing the number of estimation steps can compensate for the added variance. Reactive policy-gradient methods are already slow to converge, so increasing the number of estimation steps to compensate for adding memory is not a good solution.

The problem is quite severe in practice. Partly because the initial I-state transitions are usually close to uniform. This means that one trajectory through the world-states could invoke any trajectory through the I-state space. This requires large numbers of trials (estimation steps T) to thoroughly explore all combinations of world and I-state trajectories. In Chapter 7 we shall see that a large number of high probability I-state trajectories causes other problems. We consequently propose a fix that should also reduce the mixing time τ. Also, the next chapter introduces alternative methods of incorporating I-states into memory-less GPOMDP that are specifically designed to reduce the variance of gradient estimates.

To allow comparison of all our algorithms we defer experimental results until Chapter 8. The results will demonstrate that IState-GPOMDP can successfully learn to use memory in infinite-horizon settings, also demonstrating the effect of variance in the gradient estimates.

5.4 Summary

Key Points

I Agents implementing IState-GPOMDP interact with the world, using Monte-Carlo methods to estimate an approximation to $\nabla\eta$. IState-GPOMDP does not require knowledge of the POMDP model.

II IState-GPOMDP adds internal-state to the reactive GPOMDP algorithm of Baxter and Bartlett [2001].

III It extends the algorithms of Peshkin et al. [1999] to infinite-horizon POMDPs.

IV The discount factor β can be used to trade-off the bias and variance of IState-GPOMDPs estimates of $\nabla\eta$.

Adding I-states exacerbates the high-variance of policy-gradient methods.

Future Work

Variance reduction methods such as additive control variates [Greensmith et al., 2002], Actor-Critic methods [Konda and Tsitsiklis, 2000], and others described in Section 2.7.1, will improve IState-GPOMDP performance. We discuss some novel variance reduction methods in Chapter 9.

Rather than fix $|\mathcal{G}|$ before training we would like to dynamically grow $|\mathcal{G}|$ until the agent's performance stops improving. Chrisman [1992] and McCallum [1996] provide methods of doing this based on statistical tests over a finite amount of the observation/action history. Optimal agents can often be represented by deterministic policy-graphs [Sondik, 1978], suggesting that stochastic elements of $\omega(h|\phi, g, y)$ and $\mu(u|\theta, h, y)$ sometimes indicate that too few I-states were used. For example, stochastic I-states that transition to I-state g *or* h after observation y could be split into two I-states: one that transitions to g after observing y and one that transitions to h after observing y.

Provided Assumptions 3 and 4 of Section 3.2 are met we can build in domain knowledge into the FSC and policy to shape $\omega(h|\phi, g, y)$ and $\mu(u|\theta, h, y)$. For example, if the last n observations often reveal relevant state we can build a finite history window of length n into the agent for short-term memory and use the FSC for long-term memory. The finite history can be provided to both $\omega(h|\phi, g, y)$ and $\mu(u|\theta, h, y)$ to improve performance. Hard-wired rules can also be easily encoded in $\omega(h|\phi, g, y)$ and $\mu(u|\theta, h, y)$ to provide an initial policy.

Finally, let $\tau_{\mathcal{S}}$ be the mixing time of the world-state transitions under the worst-case fixed FSC. Similarly, let $\tau_{\mathcal{G}}$ be the mixing time of the I-state transitions under the worst-case fixed world. It would be useful to bound the global-state mixing time τ in terms of these quantities. For example, if $\tau \leq \tau_{\mathcal{S}} + \tau_{\mathcal{G}}$ then we know that the combination of a well behaved world and well behaved FSC will be well behaved together.

Algorithm 2 IState-GPOMDP

```
1: Given:
     • A parameterised class of stochastic FSCs $\{\omega(h|\phi, g, y) : \phi \in \mathbb{R}^{n_\phi}\}; g \in \mathcal{G}; h \in \mathcal{G}; y \in \mathcal{Y}$.
     • A parameterised class of stochastic policies $\{\mu(u|\theta, h, y) : \theta \in \mathbb{R}^{n_\theta}\}; h \in \mathcal{G}; y \in \mathcal{Y}$.
     • A POMDP that when controlled by $\omega(h|\phi, g, y)$ and $\mu(u|\theta, h, y)$, generates I-state and world-state trajectories distributed according to $P(\phi, \theta)$, satisfying Assumption 1 (see Section 3.2).
     • Discount factor $\beta \in [0, 1)$.
     • Arbitrary initial state $i_0$ and I-state $g_0$.
     • Observation sequence $\{y_0, y_1, \dots\}$ generated by the POMDP, I-state sequence $\{g_0, g_1, \dots\}$ generated randomly according to $\omega(\cdot|\phi, g_t, y_t)$, and action sequence $\{u_0, u_1, \dots\}$ generated randomly according to $\mu(\cdot|\theta, g_{t+1}, y_t)$.
     • Bounded reward sequence $\{r(i_1), r(i_2), \dots\}$, where $\{i_0, i_1, \dots\}$ is the (hidden) sequence of states of the world.
     • Estimation steps $T$, such that $T > \tau$.
2: Set $z_0^\phi = [0], z_0^\theta = [0], \Delta_0^\phi = [0]$, and $\Delta_0^\theta = [0]$ ($z_0^\phi, \Delta_0^\phi \in \mathbb{R}^{n_\phi}$, $z_0^\theta, \Delta_0^\theta \in \mathbb{R}^{n_\theta}$)
3: while $t < T$ do
4:   Observe $y_t$ from the world
5:   Choose $g_{t+1}$ from $\omega(\cdot|\phi, g_t, y_t)$
6:   Choose $u_t$ from $\mu(\cdot|\theta, g_{t+1}, y_t)$
7:   $z_{t+1}^\phi = \beta z_t^\phi + \frac{\nabla\omega(g_{t+1}|\phi, g_t, y_t)}{\omega(g_{t+1}|\phi, g_t, y_t)}$
8:   $z_{t+1}^\theta = \beta z_t^\theta + \frac{\nabla\mu(u_t|\theta, g_{t+1}, y_t)}{\mu(u_t|\theta, g_{t+1}, y_t)}$
9:   $\Delta_{t+1}^\phi = \Delta_t^\phi + \frac{1}{t+1}\left[r(i_{t+1})z_{t+1}^\phi - \Delta_t^\phi\right]$
10:  $\Delta_{t+1}^\theta = \Delta_t^\theta + \frac{1}{t+1}\left[r(i_{t+1})z_{t+1}^\theta - \Delta_t^\theta\right]$
11:  Issue action $u_t$
12:  $t \leftarrow t + 1$
13: end while
```

Chapter 6

Internal Belief States

I bet the human brain is a kludge.
—Marvin Minsky

So far we have assumed that the internal memory of the system consists of a finite state machine and that the agent stochastically updates its I-state at each time step. Multiple steps describe a sample trajectory through the I-state space of the finite state controller. The key idea of this section is that we do not need to sample I-states. The finite state controller is completely observable, allowing us to compute expectations over all possible I-state trajectories. At each time step we can update the probability of occupying each I-state and use this *belief* over I-states as our controller memory.

A related advantage is that probability distributions over I-states are more informative than sampled I-states. When sampling I-states we throw away information about how likely the I-state trajectory is. Passing I-state beliefs to the policy μ may permit the use of fewer I-states compared to algorithms that sample the I-state.

This section covers two methods for learning when the internal memory is represented as a belief over I-states. The first approach uses hidden Markov models to uncover hidden state. This approach failed to control POMDPs as well as IState-GPOMDP, even given an infinite number of HMM states. The essential problem, to be demonstrated in detail, is that learning an HMM that models the world well does not imply that the HMM can influence the long-term reward. This initial failure prompted a shift away from HMM methods to the second method which we view as a partial Rao-Blackwellisation of IState-GPOMDP.

6.1 Hidden Markov Models for Policy Improvement

Because partial observability hides the true state of the world, we can use existing algorithms that reveal hidden state to improve the performance of POMDP methods. Hidden Markov models (HMMs) are a candidate that have previously been used in conjunction with value-function methods and, more recently, policy-gradient methods. Appendix D introduces the basic HMM background required for understanding this chapter. HMMs are a generative model, that is, they model stochastic sequences without being driven by the available data. In the POMDP setting we may wish to model the sequence of observations y_t, or actions u_t, or, as we shall see in this Section, rewards r_{t+1}; but the predictions should be driven by the available information, such as the last observation and action.

Bengio and Frasconi [1996] describe an extension to HMMs, called Input/Output HMMs, that can model one sequence while being driven by a related sequence. The idea is that the driving sequence contains information useful to prediction of the sequence being modeled. More recently, statistical models driven by observations have been studied under the name *conditional random fields* [Lafferty et al., 2001, McCallum et al., 2000]. Appendix D.2 describes our simple implementation of IOHMMs. The message of this chapter only requires understanding that IOHMMs are a straightforward generalisation of HMMs that allow an external sequence to drive transitions.

IOHMMs have been used to solve small POMDPs where actions drive the IOHMM state transitions and the IOHMM predicts the observations y_t [Chrisman, 1992, McCallum, 1996]. HMMs can also be layered at different time scales for hierarchical reinforcement learning [Theocharous et al., 2000]. An alternative scheme, which was implemented using policy-gradient methods, *drives* IOHMM-state transitions with observations y_t then *generates* actions u_t [Shelton, 2001b,a].

6.1.1 Predicting Rewards

A problem with previous IOHMM methods is that estimation of the hidden state ignores the most useful indicator of policy performance: the reward. Predicting rewards reveals the hidden state relevant to predicting policy performance, and that may consequently be relevant to choosing actions that lead to high rewards.

To include the reward we initially investigated using the IOHMM to predict rewards instead of observations. The IOHMM-state transition probabilities are driven by the observations and, optionally, the actions. The idea is that the hidden state revealed by predicting rewards is the most relevant to maximising the long-term average reward η. The remainder of Section 6.1 introduces this approach and describes the drawbacks of this algorithm that motivate the approach of the next section.

6.1.2 The IOHMM-GPOMDP Algorithm

We first describe the step-by-step operation of the algorithm, assuming training is complete. Then we describe how the agent is trained.

6.1.2.1 Augmenting Policies with IOHMMs

The IOHMM

The function of $\omega(h|\phi, g, y)$ is now performed by an IOHMM, as shown in Figure 6.1. The I-states are simply the IOHMM states. There are 3 IOHMM-states in Figure 6.1. We compute and store the probability of being in each IOHMM-state at the current time. This vector of IOHMM-state probabilities at time t is denoted by α_t. It tells us the probability of each IOHMM-state given the observation history $\bar{y}_{t-1} = \{y_0, \dots, y_{t-1}\}$, the observed reward sequence $\bar{r}_{t-1} = \{r_1, \dots, r_{t-1}\}$, and the IOHMM parameters ϕ. The dependencies are only up to time $t-1$ because of the convention that α_t is the internal-belief at the start of a time step, prior to receiving observation y_t. In HMM terminology the vector α_t is known as the *forward probability* of $\bar{r}_{t-1}$. We will usually refer to α_t as the *I-state belief.*

In Section 2.2 we defined $\mathcal{G}$ to be the set of I-states. For the FSCs considered in previous chapters there is a one-to-one correspondence between the number of I-states and the number of states in the FSC. Now, because α is a vector of real numbers, $\mathcal{G}$ is infinite under the original definition. For convenience we slightly alter the definition of $\mathcal{G}$ in the context of internal belief states to mean the number of nodes in the internal system, that is, the number of IOHMM-states. For example, the IOHMM shown in Figure 6.1 has 3 IOHMM-states so $|\mathcal{G}| = 3$. This satisfies the intuition that an FSC with $|\mathcal{G}|$ I-states has similar architectural complexity to an IOHMM with $|\mathcal{G}|$ IOHMM-states.

An IOHMM consists of IOHMM-state transition probabilities, denoted $\omega(h|\phi_\iota, g, y)$; and IOHMM-state reward emission probabilities, denoted by $\xi(r|\phi_\xi, g)$. The use of these two functions is illustrated in Figure 6.1. The I-state update parameters split into two subsets $\phi = [\phi_\iota, \phi_\xi]$. The parameter subset ϕ_ι controls how the *input* drives the transitions. The parameter subset ϕ_ξ describes how the *output* is generated. The IOHMM-state transition probability function $\omega(h|\phi_\iota, g, y)$ is exactly the same as the FSC I-state transition probability function $\omega(h|\phi, g, y)$ except that only the subset ϕ_ι of ϕ is used to compute the transition probabilities. If we used normal HMMs instead of IOHMMs, $\omega(h|\phi_\iota, g, y)$ would have no dependence on y.

Recall that $\bar{y}_t$ is the history of observations up to time t, $\bar{r}_t$ is the history of rewards up to time t, r_t is the scalar instantaneous reward for time t and h is the proposed next I-state. With these definitions the IOHMM-state belief (equivalent to the I-state belief) update is

$$\alpha_{t+1}(h|\phi, \bar{y}_t, \bar{r}_t) = \sum_{g \in \mathcal{G}} \alpha_t(g|\phi, \bar{y}_{t-1}, \bar{r}_{t-1}) \xi(r_t|\phi_\xi, g) \omega(h|\phi_\iota, g, y_t). \tag{6.1}$$

This equation computes the probability of I-state h at time $t+1$ by computing the sum of probabilities of $g \to h$ over all possible previous I-states g. The probability of emitting the observed reward r_t in each g is also factored into the update. The value of the vector α_t generally vanishes over time, so practical IOHMM algorithms

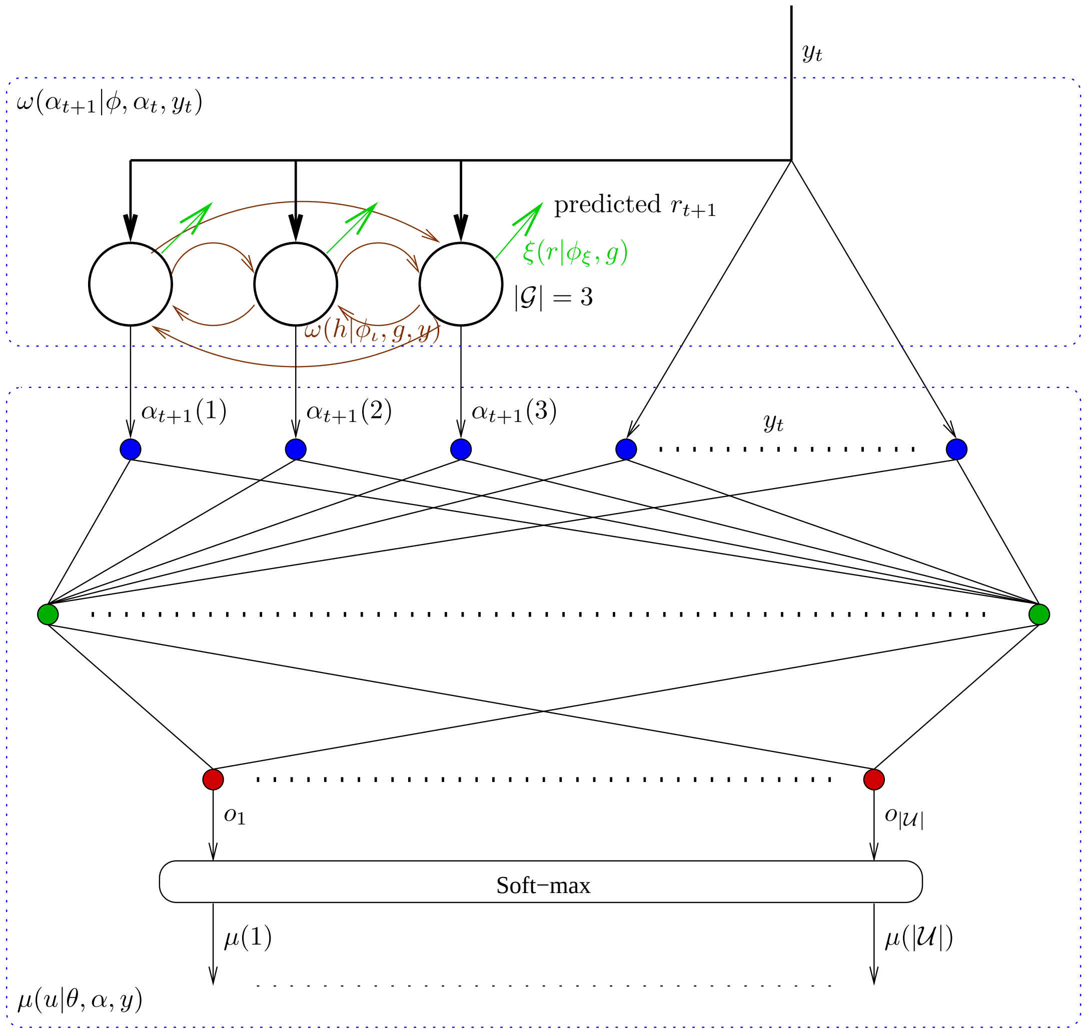


Figure 6.1: The architecture of the IOHMM-GPOMDP algorithm. The observation is passed to both the IOHMM and the policy $\mu(u|\theta, h, y)$. The policy is shown here as an artificial neural network (ANN). During each step the first phase is to compute α_{t+1} from α_t, y_t, and r_t. Then α_{t+1} and y_t are passed to the ANN that outputs a vector of real numbers. The output vector is transformed into a probability distribution over actions by the soft-max function.

must re-normalise at each step. Methods of doing this in the context of HMMs are well studied [Rabiner, 1989] and will not be considered further. The update implies that there is no stochastic component for the update of the internal state information, that is, the I-state belief update is deterministic. Compare this to the IState-GPOMDP update that *samples* an I-state h given the current I-state and observation.

An alternative definition of Equation (6.1), using more traditional HMM terminology, can be found in Appendix D.2. The definition here is slightly unusual because it computes the reward emission probability of a state *before* making a transition. This is because the quantity we are modelling, r_{t+1}, is not received until *after* computing α_t and issuing action u_t. The first time r_{t+1} can be used to update α is the start of the next step, when it is labelled r_t.

To parameterise $\omega(h|\phi_\iota, g, y)$ we use a table with one probability for each combination of (g, h, y), as described in Appendix D.2. The probabilities summed over all next states h, given g and y, sum to one. This scheme requires $\mathcal{Y}$ to be relatively small.

We parameterise $\xi(r|\phi_\xi, g)$ using tables in one experiment, and a single Gaussian in another. These are both standard methods for parameterising HMM and IOHMM emission probabilities. Using a table for to parameterise $\xi(r|\phi_\xi, g)$ implies there is a finite set of reward values that the POMDP can issue. Gaussian parameterisations can model continuous distributions of rewards.

The Policy

The policy $\mu(u|\theta, \alpha, y)$ computes a distribution over actions given the I-state belief and the observation. Because the I-state belief is a vector of real numbers, we must use function approximation to evaluate $\mu(u|\theta, \alpha, y)$. We use artificial neural networks (ANNs). The internal belief α_t is passed to the ANN using $|\mathcal{G}|$ inputs, while the other inputs encode y_t. Our IOHMM-GPOMDP experiments all have small finite $\mathcal{Y}$, so we use 1-in-n encoding for presentation of y_t to the ANN.

Once the distribution has been evaluated, a u is sampled from the distribution and issued as action u_t. The details of computing the gradient of $\mu(u_t|\theta, \alpha_{t+1}, y_t)$ are given in Section 3.4.2. Then the world-state is updated and reward r_{t+1} is received to end the step. The reward is used in Equation (6.1) to compute α_{t+1} during the next step.

6.1.2.2 Training Using IOHMM-GPOMDP

Algorithm 3 describes the training process. It interleaves phases of IOHMM training (see Appendix D.2) with memory-less IState-GPOMDP (see Algorithm 2). We refer to memory-less IState-GPOMDP as just GPOMDP. Lines 3–13 execute the current policy, storing all observations[1] and rewards. In line 14, the stored history, that is T_{iohmm} steps long, is used to update the IOHMM parameters. We have deliberately left the training details of the IOHMM unspecified because there are many well established training procedures. The novel contribution of the IOHMM-GPOMDP algorithm is to use IOHMMs for predicting rewards. We assume that the chosen training procedure will maximise the likelihood of the reward sequence $\bar{r}$ given the observation sequence. An IOHMM training phase ends when consecutive IOHMM iterations do not increase the likelihood of $\bar{r}$.

[1]We assume that observations can always be augmented with the last action if actions should also be taken into account.

Algorithm 3 IOHMM-GPOMDP

```
1: Given:
   • Parameterised class of randomised policies $\{\mu(u|\theta,\alpha,y) : \theta \in \mathbb{R}^{n_\theta}\}; \alpha \in [0,1]^{|\mathcal{G}|}; y \in \mathcal{Y}$.
   • An IOHMM with $|\mathcal{G}|$ IOHMM-states, driving inputs $y_t \in \mathcal{Y}$, emission symbols in $[-R, R]$, transition probabilities $\omega(h|\phi_\iota, g, y)$, reward emission probabilities $\xi(r|\phi_\xi, g)$, all parameterised by $\phi = [\phi_\iota, \phi_\xi]$ initialised to small random values.
   • Arbitrary initial world state $i_0 \in \mathcal{S}$ and uniform internal belief $\alpha_0 \in \mathbb{R}^{|\mathcal{G}|}$.
   • Observation sequence $y_0, y_1, \dots$ generated by the POMDP, I-state belief sequence $\alpha_0, \alpha_1, \dots$ generated deterministically according to (6.1), and action sequence $u_0, u_1, \dots$ generated randomly according to $\mu(\cdot|\theta, \alpha_{t+1}, y_t)$.
   • $[-R, R]$ bounded reward sequence $r_1 = r(i_1), r_2 = r(i_2), \dots$, where $i_1, i_2, \dots$ is the hidden sequence of states of the environment.
   • IOHMM sample length $T_{\text{iohmm}}$.
   • IState-GPOMDP estimation length $T_{\text{grad}}$; discount $\beta \in [0, 1)$; step size $\gamma > 0$.
2: while $\|\Delta_{T_{\text{grad}}}\| > \epsilon$ do
3:   $t = 0$
4:   $\bar{y} = \bar{r} = \emptyset$
5:   while $t < T_{\text{iohmm}}$ do
6:     Observe $y_t$
7:     Append $y_t$ to $\bar{y}$
8:     $\alpha_{t+1}(h|\phi, \bar{y}_t, \bar{r}_t) = \sum_{g\in\mathcal{G}} \alpha_t(g|\phi, \bar{y}_{t-1}, \bar{r}_{t-1})\xi(r_t|\phi_\xi, g)\omega(h|\phi_\iota, g, y_t)$
9:     Choose $u_t$ from $\mu(\cdot|\theta, \alpha_{t+1}, y_t)$
10:    Receive $r_{t+1}$
11:    Append $r_{t+1}$ to $\bar{r}$
12:    $t \leftarrow t + 1$
13:  end while
14:  train_IOHMM_to_convergence($\phi$ ,$\bar{y}$, $\bar{r}$)
15:  Set $z_0 = 0$, and $\Delta_0 = 0$ $(z_0, \Delta_0 \in \mathbb{R}^{n_\theta})$
16:  $t = 0$
17:  while $t < T_{\text{grad}}$ do
18:    Observe $y_t$
19:    $\alpha_{t+1}(h|\phi, \bar{y}_t, \bar{r}_t) = \sum_{g\in\mathcal{G}} \alpha_t(g|\phi, \bar{y}_{t-1}, \bar{r}_{t-1})\xi(r_t|\phi_\xi, g)\omega(h|\phi_\iota, g, y_t)$
20:    Choose $u_t$ from $\mu(\cdot|\theta, \alpha_{t+1}, y_t)$
21:    $z_{t+1} = \beta z_t + \frac{\nabla\bar{\mu}(u_t|\theta,\alpha_{t+1},y_t)}{\bar{\mu}(u_t|\theta,\alpha_{t+1},y_t)}$
22:    $\Delta_{t+1} = \Delta_t + \frac{1}{t+1}[r_{t+1}z_{t+1} - \Delta_t]$
23:    $t \leftarrow t + 1$
24:  end while
25:  $\theta \leftarrow \theta + \gamma\Delta_{T_{\text{grad}}}$
26: end while
```

Lines 15–24 use the newly estimated IOHMM parameters to compute an estimate of $\nabla\eta$ with respect to the parameters θ. The observations y_t for GPOMDP are augmented by the internal state α_t as provided by the IOHMM.

Finally, in line 25, the gradient estimate is used to update the parameters θ, completing the cycle of independently updating ϕ then θ. In practice we do not fix the step size γ, but use the conjugate-gradient procedure outlined in Section 3.5.

Our IOHMM implementation fixes $|\mathcal{G}|$ before training. Methods that could be applied to automatically grow $|\mathcal{G}|$ during training are discussed by Chrisman [1992] and McCallum [1996]. Such methods keep statistics about how successful each IOHMM state is at predicting rewards. If the prediction for a state is unreliable, or the output reward distribution is multi-modal, it indicates that the state may need to be split.

If rewards can take a wide number of values in $[R, -R]$, then continuous emission distributions such as mixtures of Gaussians [Rabiner, 1989] for each state are preferable. Using a single Gaussian per state to model rewards is equivalent to the assumption that each state's emission distribution is peaked around a single reward. If there are only a small number of instantaneous reward values then discrete output IOHMMs can also be used. As suggested by Rabiner [1989], we found it important to use good initialisations of the emission distributions in order to obtain IOHMM convergence to a good maximum. Good initialisations are often easy to devise using the assumption that each IOHMM-state models a particular POMDP reward value.

6.1.3 Convergence of the IOHMM-GPOMDP Algorithm

Line 14 of Algorithm 3 runs IOHMM training until the ϕ parameters converge to a local maximum.[2] If the next gradient estimation phase for the policy produces a near zero estimate then the θ parameters will not change. When this happens we declare the whole system converged since the IOHMM ϕ parameters and the policy θ parameters are both at local maxima. Barring the effects of using different samples, neither the ϕ or θ can change during future training.

Unfortunately, the re-estimation of the IOHMM parameters in line 14 can lead to an arbitrarily large decrease in the performance of the policy. Consider a world with a straight road 5 sections long and assume the optimal policy is to move from one end to the other (essentially the Load/Unload problem of Figure 2.2(a)). Initially both the IOHMM and $\mu(u|\theta, \alpha, y)$ are initialised with random parameters. The following can occur:

1. With random $\mu(u|\theta, \alpha, y)$ the IOHMM learns that rewards are more likely when

[2]Baum-Welch training *or* gradient ascent training for HMMs and IOHMMs converges to a local maximum of the likelihood function.

`move left` actions occur after the "`L`" observation, or `move right` actions occur after the "`U`" observation.

2. μ becomes a good policy based on the current IOHMM, obtaining rewards every 5 steps.

3. The change in policy parameters θ, changes the extrema of the IOHMM. Consequently the IOHMM can move into a different local maximum based on the sample $\bar{y}$ gathered using the old ϕ. The IOHMM may learn that a reward usually occurs every 5 steps.

4. The policy represented by θ is no longer optimal under the IOHMM represented by the new ϕ. The reward may drop to a level worse than the uniform random policy.

5. The policy is no longer near optimal and rewards no longer occur every 5 steps, thus the IOHMM must re-learn the concept it learnt in the first step.

Thus, allowing the IOHMM and $\mu(u|\theta, \alpha, y)$ to bootstrap from each other can result in learning cycles that do not converge. The fundamental reason is that the IOHMM maximises the likelihood of $\bar{r}$, but GPOMDP maximises a different criterion: the long-term average reward η. If, during Step 4, GPOMDP quickly learns a new policy that is good under the new ϕ, then we may achieve convergence. However we do not usually want GPOMDP to completely converge in one iteration because it will generally end up in a poor maximum.

6.1.4 Drawbacks of the IOHMM-GPOMDP Algorithm

We have just described how IOHMM-GPOMDP may fail to converge. A second drawback is that being able to successfully predict rewards does not necessarily reveal the hidden state necessary for optimal policies. Consider the POMDP shown in Figure 6.2(a). Solid lines are deterministic transitions. Dotted lines are followed with probability 0.5. Actions only impact rewards when the current observation is `a`. To chose the optimal action we must recall one past observation. If we observe the sequence `da`, then execute action `u2`; if we observe `ea`, then execute `u1`. This is a simple policy requiring $|\mathcal{G}| = 2$ and it is easily learnt by the IState-GPOMDP algorithm of Section 2. Now consider the task of learning an IOHMM to predict rewards. Figure 6.2(b) shows an optimal IOHMM with $|\mathcal{G}| = 3$. This IOHMM waits for observation `c` or `b` while predicting 0 reward. Observation `c` always implies a reward of -1 after the next action and `b` always implies a reward of 1 after the next action. This IOHMM successfully reveals the hidden state associated with observation `r` because the IOHMM forward

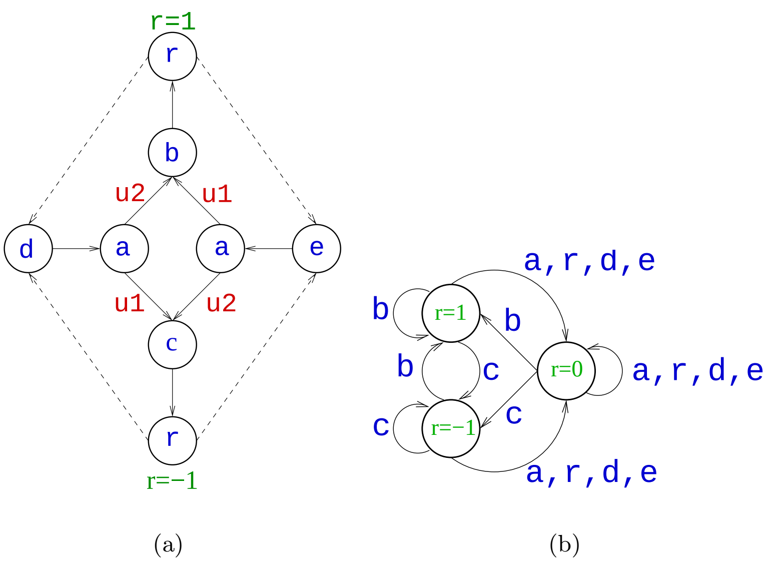


Figure 6.2: 6.2(a) A POMDP with $\mathcal{U} = \{\texttt{u1},\ \texttt{u2}\}$ and $\mathcal{Y} = \{\texttt{a,b,c,d,e,r}\}$. The states with observation `r` have non-zero rewards shown. This POMDP can always be solved by IState-GPOMDP with $|\mathcal{G}| = 2$ but will not be solved by the *optimal* reward prediction IOHMM shown in Figure 6.2(b), or any IOHMM with more or less states.

probability will indicate which of the two `r` world states we are in. However, it does not allow us to implement an optimal policy because we need to know whether to issue action `u1` or `u2` *before* seeing observation `c` or `b`. Adding states to the IOHMM does not help because the IOHMM is not required to remember the critical observations `d` and `e` in order to optimise its performance.

In short, IOHMM-GPOMDP suffers from the same difficulties as value-function methods: the agent can learn to predict rewards well while failing to learn a good policy. These two drawbacks cause poor performance of IOHMM-GPOMDP on non-trivial problems, which we will observe in Chapter 8.

Other configurations of HMMs are possible. For example, the HMM can predict world states as well as rewards, or observations and rewards. The state transitions $\omega(h|\phi_\iota, g, y_t)$ could also be driven by actions. Alternative HMMs might provide optimal performance but still suffer from a lack of direct optimisation of the long-term reward. One consequence of indirect optimisation is that many more HMM states than FSC states might be necessary. The HMM might have to explicitly model all world states to provide sufficient information to the agent, but an FSC driven by maximising the reward can filter out information irrelevant to the long-term reward.

6.2 The Exp-GPOMDP Algorithm

One reason for the high variance of existing policy-gradient methods is the noise introduced through estimating the gradient by sampling a trajectory through the environment state space.[3] In Chapter 5 we viewed the I-states as augmenting the world-state space, resulting in the method of sampling trajectories through both world-states and I-states. However, we can do better than simply sampling one long I-state trajectory. Recall the reasons for introducing sampling in the first place:

1. The POMDP model may not be available;

2. dynamic programming becomes infeasible for large $|\mathcal{S}|$.

Since I-state transitions are controlled by the known function $\omega(h|\phi, g, y)$, and the number of I-states is typically small compared to the world-state space, the main reasons for using simulation do not apply. In short, we can use the FSC model $\omega(h|\phi, g, y)$ to compute the expectation of the gradient over *all possible I-state trajectories.*

The Exp-GPOMDP algorithm shown in Algorithm 4 is a partly Rao-Blackwellised version of IState-GPOMDP; computing the expectation over I-state trajectories but not world-state trajectories. Rao-Blackwellisation of Monte-Carlo estimators reduces their variance [Casella and Robert, 1996].

We proceed by replacing $\mu(u_t|\theta, g_{t+1}, y_t)$ with $\bar{\mu}(u_t|\phi, \theta, \bar{y}_t)$ which parameterises action probabilities based on the current I-state belief and observation. The I-state update is still parameterised by a stochastic FSC described by ϕ, but the update to the I-state *belief* is not stochastic. Let $\alpha_t(g|\phi, \bar{y}_{t-1})$ be the probability that $g_t = g$ given the current parameters and all previous observations. The recursive update for the I-state belief is

$$\alpha_{t+1}(h|\phi, \bar{y}_t) = \sum_{g \in \mathcal{G}} \alpha_t(g|\phi, \bar{y}_{t-1}) \omega(h|\phi, g, y). \tag{6.2}$$

This is similar to the forward probability update given by Equation (6.1) but contains no component modelling emission likelihoods (so no normalisation is required). We have written $\alpha_t(g|\phi, \bar{y}_{t-1})$ with an explicit dependence on all past observations $\bar{y}_{t-1}$. We do not explicitly store all past observations because the Markov nature of the process mean α_t and y_t summarise all the information necessary to update the belief to α_{t+1}. In other words, the dependence on $\bar{y}_t$ could be replaced by (α_{t-1}, y_t), but we use the former for brevity.

[3]The GAMP algorithm of Section 4.2 is an exception since it avoids simulating trajectories by direct gradient estimation from the POMDP model.

The new form of the policy that takes the expectation of μ over internal states is

$$\bar{\mu}(u|\phi,\theta,\bar{y}_t) := \sum_{h\in\mathcal{G}} \alpha_{t+1}(h|\phi,\bar{y}_t)\mu(u|\theta,h,y_t). \tag{6.3}$$

We can view these equations as representing a form of IOHMM, where $\bar{\mu}$ is the probability of emitting symbol (action) u given all previous observations. To finish the algorithm we need to compute $\frac{\nabla\bar{\mu}}{\bar{\mu}}$ for the choice of u_t. We use it in the IState-GPOMDP algorithm in place of $\frac{\nabla\mu}{\mu}$. This is straight forward after noting that, for each h, $\nabla\alpha_{t+1}(h|\phi,\bar{y}_t)$ can be updated recursively from all $\nabla\alpha_t(g|\phi,\bar{y}_{t-1})$ by applying the product rule to Equation (6.2), as shown in line 10 of Algorithm 4. The gradient of $\bar{\mu}$ is split into the ϕ and θ components

$$\frac{\nabla\bar{\mu}}{\bar{\mu}} = \frac{\left[\nabla^{\phi}\bar{\mu}, \nabla^{\theta}\bar{\mu}\right]}{\bar{\mu}},$$

where the ϕ components are

$$\begin{aligned}\frac{\nabla^{\phi}\bar{\mu}}{\bar{\mu}} &= \frac{\left[\frac{\partial\bar{\mu}}{\partial\phi_1},\dots,\frac{\partial\bar{\mu}}{\partial\phi_{n_\phi}}\right]}{\bar{\mu}}\\ &= \frac{\sum_{g\in\mathcal{G}}\left(\nabla\alpha_{t+1}(h|\phi,\bar{y}_t)\right)\mu(u|\theta,h,y_t)}{\sum_{g\in\mathcal{G}}\alpha_{t+1}(h|\phi,\bar{y}_t)\mu(u|\theta,h,y_t)},\end{aligned}$$

and the θ components are

$$\begin{aligned}\frac{\nabla^{\theta}\bar{\mu}}{\bar{\mu}} &= \frac{\left[\frac{\partial\bar{\mu}}{\partial\theta_1},\dots,\frac{\partial\bar{\mu}}{\partial\theta_{n_\theta}}\right]}{\bar{\mu}}\\ &= \frac{\sum_{g\in\mathcal{G}}\alpha_{t+1}(g|\phi,\bar{y}_t)\nabla\mu(u|\theta,g,y_t)}{\sum_{g\in\mathcal{G}}\alpha_{t+1}(g|\phi,\bar{y}_t)\mu(u|\theta,g,y_t)}.\end{aligned}$$

Another way to think of this algorithm is as resembling IState-GPOMDP, but with no explicity memory process $\omega(h|\phi,g,y)$. Instead, we embed a deterministically updated memory vector directly into the policy. The whole process is still Markov because the next global-state (including the current internal-belief), is only a stochastic function of the current global-state.

6.2.1 Complexity of Exp-GPOMDP

The combination of Equations (6.2) and (6.3) imply that each step now has a complexity of $O(|\mathcal{G}||\mathcal{U}| + |\mathcal{G}|^2)$ if we are using lookup-tables for $\omega(h|\phi,g,y)$ and $\mu(u|\theta,h,y)$. Recall from Section 5.3.1 that IState-GPOMDP has a complexity of $O(|\mathcal{U}| + |\mathcal{G}|)$ per step for the same parameterisation. The complexity can be reduced by using alternative parameterisations, but the summations over the I-states in Equations (6.2) and (6.3)

Algorithm 4 Exp-GPOMDP

```
1: Given:
     • Same requirements as IState-GPOMDP (Algorithm 2).
     • The joint process $\{\alpha_t \in \mathbb{R}^{|\mathcal{G}|}, i_t \in \mathcal{S}\}$ is ergodic.
2: Set $z_0 = [z_0^\phi, z_0^\theta] = [0]$, and $\Delta_0 = [\Delta_0^\phi, \Delta_0^\theta] = [0]$ $(z_0, \Delta_t \in \mathbb{R}^{n_\phi + n_\theta} \quad \forall t)$.
3: for each internal state $g$ do
4:   Set $\alpha_0(g) = 1/|\mathcal{G}|$ and
     $\nabla\alpha_0(g) = [\nabla^\phi \alpha_0(g), \nabla^\phi \alpha_0(g)] = [0]$ $(\nabla\alpha_t(g) \in \mathbb{R}^{n_\phi + n_\theta} \quad \forall t)$.
5: end for
6: while $t < T$ do
7:   Observe $y_t$ from the world.
8:   for each internal state $h$ do
9:     $\alpha_{t+1}(h|\phi, \bar{y}_t) = \sum_{g \in \mathcal{G}} \alpha_t(g|\phi, \bar{y}_{t-1}) \omega(h|\phi, g, y_t)$
10:    $\nabla\alpha_{t+1}(h|\phi, \bar{y}_t) =$
       $\sum_{g \in \mathcal{G}} (\nabla\alpha_t(g|\phi, \bar{y}_{t-1}) \omega(h|\phi, g, y_t)) + \alpha_t(g|\phi, \bar{y}_{t-1}) \nabla\omega(h|\phi, g, y_t)$
11:  end for
12:  Choose $u_t$ from $\bar{\mu}(u|\theta, \phi, \bar{y}_t) = \sum_{g \in \mathcal{G}} \alpha_{t+1}(h|\phi, \bar{y}_t) \mu(u|\theta, h, y_t)$
13:  $z_{t+1} = \beta z_t + \frac{\nabla\bar{\mu}(u_t|\theta,\phi,\bar{y}_t)}{\bar{\mu}(u_t|\theta,\phi,\bar{y}_t)}$
14:  $\Delta_{t+1} = \Delta_t + \frac{1}{t+1} [r(i_{t+1}) z_{t+1} - \Delta_t]$
15:  Issue action $u_t$
16:  $t \leftarrow t + 1$
17: end while
```

mean that Exp-GPOMDP will always have greater complexity than IState-GPOMDP by a factor of $|\mathcal{G}|$. The memory use of Exp-GPOMDP is also higher due to the need to store the extra set of gradients $\nabla\alpha_t(g|\phi, \bar{y}_{t-1})$ for each g.

Experimental results in Section 8 show that while the wall clock time of Exp-GPOMDP may be greater than IState-GPOMDP, Exp-GPOMDP requires fewer simulation steps, which is desirable when world interactions are expensive. This provides evidence that taking the expectation over I-states partially alleviates the problem of increased variance due to addition of I-states. See Section 5.3.2 for the original discussion.

Finally, we believe the Exp-GPOMDP gradient estimate has the same convergence guarantees as IState-GPOMDP (see Theorem 4, Section 5.2). Specifically, in the limit as the number of estimation steps $T \to \infty$, the gradient estimate converges to $\pi'(\nabla P)J_\beta$. At the current time the proof exists in sketch form [Baxter, 2002]. The outline of the proof is the same as that for IState-GPOMDP (see the proof of Theorem 4, Appendix A.2.2). However, the details are more complex for Exp-GPOMDP because the internal-state belief α_t is uncountably infinite. The increased complexity is reflected by the extra assumption in Algorithm 4 that the joint process $\{\alpha_t, i_t\}$ is ergodic.[4]

6.2.2 An Alternative Exp-GPOMDP Algorithm

There may be alternative ways to Rao-Blackwellise Exp-GPOMDP that result in new algorithms. For example, an algorithm that tracks the I-state belief, but instead of generating actions based on the expectation over all I-states (6.3), we could sample an I-state g_{t+1} from $\alpha_{t+1}(\cdot|\phi, \bar{y}_t)$, and then sample the action distribution from $\mu(u_t|\theta, g_{t+1}, y_t)$ only. Using such a scheme, the execution of a single step at time t might be:

1. update the I-state belief using (6.2);
2. sample g_{t+1} from $\alpha_{t+1}(\cdot|\phi, \bar{y})$;
3. sample u_t from $\mu(\cdot|\theta, g_{t+1}, y_t)$;
4. compute the gradient contributions for the eligibility trace.

Suppose at time t we sample I-state g_{t+1} from α_{t+1}, then the algorithm would compute the expectation over all I-state trajectories that lead to I-state g_{t+1}, instead of over all I-state trajectories. Such an algorithm would be useful due to its lower per-step complexity than Exp-GPOMDP, while hopefully still having lower variance than IState-GPOMDP. Because the alternative algorithm makes partial use of α_t, we plotted it half way up the I-state axis of Figure 3.2. The derivation and convergence properties of such algorithms need to be investigated.

6.2.3 Related Work

Using a recursively updated forward probability variable α to compute an expectation over all possible paths through a state lattice is an important aspect of hidden Markov Model training [Rabiner, 1989]. By viewing state transitions as being driven by the observations, and viewing actions as symbols generated by the HMM, Exp-GPOMDP becomes a method for training Input/Output HMMs without using the backward probability component of HMM training. An important difference compared to HMM training is that Exp-GPOMDP does not seek to maximise the conditional likelihood of any particular sequence.

Exp-GPOMDP is similar to the finite-horizon algorithm presented by Shelton [2001b], which is a gradient-ascent HMM algorithm where emissions are actions. In that paper the HMM backward probability is used as well as the forward probability α. In the

[4]This assumption is probably automatically satisfied under the existing assumption that the process $\{g_t, i_t\}$ is ergodic (see Assumption 1, Section 3.2), but this is yet to be proven.

infinite-horizon setting there is no natural point to begin the backward probability calculation so our algorithm uses only α.

6.3 Summary

Key Points

I IOHMM-GPOMDP uses IOHMMs to predict rewards, attempting to reveal the hidden state relevant to predicting rewards. Memory-less IState-GPOMDP then learns a policy based on the IOHMM-state belief and the current observations.

II IOHMM-GPOMDP is not guaranteed to converge to a local maximum of the long-term reward η, and the IOHMM may not reveal enough hidden state to allow the best policy to be learnt.

III The I-state model is given by the known function $\omega(h|\phi, g, y)$, thus we can compute the expectation over I-state trajectories.

IV Exp-GPOMDP does this, reducing the variance of the estimate.

V Exp-GPOMDP complexity scales quadratically with the number of I-states, preventing large numbers of I-states being used.

Future Work

Further analysis is required to determine POMDPs for which the IOHMM-GPOMDP method will converge and work better than Exp-GPOMDP. The further work discussion for IState-GPOMDP also applies to Exp-GPOMDP, that is, many variance reduction methods from the literature can be applied. The alternative Exp-GPOMDP algorithm needs to be implemented and tested on the scenarios described in Chapter 8.

We compute $\bar{\mu}(\cdot|\phi, \theta, \bar{y}_t)$ by taking the expectation of $\mu(u|\theta, h, y)$ over all I-states for each action. We could implement $\bar{\mu}(\cdot|\theta, \phi, \bar{y}_t)$ directly, using function approximation. For example, we could build a neural network implementing $\bar{\mu}(\cdot|\phi, \theta, \bar{y}_t)$, propagating gradients back to the α inputs to compute derivatives with respect to ϕ. This could result in improved policies because the action choice can take into account features such as the relative probability of each I-state. We apply a more direct implementation of $\bar{\mu}(u|\phi, \theta, \bar{y}_t)$ for the speech recognition experiments of Section 10.3.

Finally, we have claimed without proof that Exp-GPOMDP has lower variance than IState-GPOMDP. Casella and Robert [1996] proves that Rao-Blackwellisation applied to Monte-Carlo methods has a variance reducing effect. We hope to use similar proof methods to provide theoretical guarantees about the degree of variance reduction achieved by using Exp-GPOMDP instead of IState-GPOMDP. Chapter 8 provides empirical evidence for the variance reduction.

Chapter 7

Small FSC Gradients

If an algorithm is going to fail, it should have the decency to quit soon.

—Gene Golub

Previous advocates of direct search in the space of FSCs [Meuleau et al., 1999a,b, Peshkin et al., 1999, Lanzi, 2000] report success, but only on POMDPs with a few tens-of-states that only need a few bits of memory to solve. The Load/Unload problem (see Figure 2.2(a)) is a common example. Meuleau et al. [1999a] and Lanzi [2000] comment briefly on the difficulties experienced with larger POMDPs. In this chapter we analyse one reason for these difficulties in the policy-gradient setting. Our analysis suggests a trick that allows us to scale FSCs to more interesting POMDPs, as demonstrated in Chapter 8.

7.1 Zero Gradient Regions of FSCs

In our early experiments we observed that policy-gradient FSC methods initialised with *small random parameter values* failed to learn to use the I-states for non-trivial scenarios. This was because the gradient of the average reward η, with respect to the I-state transition parameters ϕ, was too small. Increasing $\max_l |\phi_l|$ (the range of the random number generator) helps somewhat, but increasing this value too much results in the agent starting near a local maximum because the soft-max function is saturated.

The fundamental cause of this problem comes about because, with small random parameters, the I-state transition probabilities are close to uniform. This means the I-states are essentially undifferentiated. If, in addition to the I-state transitions being close to uniform, the action probabilities are similar for each I-state g, then varying the trajectory through I-states will not substantially affect the reward. The net result is that the gradient of the average reward with respect to the I-state transition parameters will be close to 0. Hence, policies whose starting distributions are close to uniform will be close to a point of 0 gradient with respect to the internal-state parameters, and will tend to exhibit poor behaviour in gradient-based optimisation. The following theorem formalises this argument.

Theorem 6. *If we choose θ and ϕ such that $\omega(h|\phi,g,y) = \omega(h|\phi,g',y)\ \forall h,g,g',y$ and $\mu(u|\theta,h,y) = \mu(u|\theta,h',y)\ \forall u,h,h',y$ then $\nabla^{\phi}\eta = [0]$.*

This theorem is proved in Appendix A.3. Even if the conditions of the theorem are met when we begin training, they may be violated by updates to the parameters θ during training, allowing a useful finite state controller to be learnt. Appendix A.3 also analyses this situation, establishing an additional condition for *perpetual* failure to learn a finite state controller, despite changes to θ during learning. The additional condition for lookup tables is $\omega(h|\phi,g,y) = 1/|\mathcal{G}| \quad \forall g,h,y$, which is satisfied by a table initialised to a constant value, such as 0.

The same problem has been observed in the setting of learning value functions when the policy can set memory bits [Lanzi, 2000]. Multiple trajectories through the memory states have the same reward, which Lanzi calls *aliasing on the payoffs*. A solution was hinted at by Meuleau et al. [1999a] where it was noted that finite state controllers were difficult to learn using Monte-Carlo approaches unless strong structural constraints were imposed. Imposing structural constraints without using domain knowledge is the essence of our proposed solution to small FSC gradients.

Recurrent neural networks (RNNs) provide an alternative memory mechanism that is described in Section 2.6.2.3. Like FSC agents, the RNN can be decomposed into a component that makes internal-state transitions and a component that chooses actions. However, the internal state is a vector of real numbers rather than an element from a finite set. This introduces its own problems, such as internal-state gradients that vanish or explode [Hochreiter and Schmidhuber, 1997], resulting in implementations that can handle no more I-states than FSC agents [Lin and Mitchell, 1992].[1]

One interesting exception to the poor performance of FSC methods is Glickman and Sycara [2001], where an Elman network — an RNN where all outputs are fed back to the hidden layer — parameterises an agent for the New York Driving scenario. This POMDP has over 21,000 states and requires multiple bits of memory. Surprisingly, Elman networks trained using an evolutionary algorithm outperformed the UTREE algorithm (see Section 2.6.2.2). The best performance was achieved using hidden units that emitted 1 or 0 stochastically. Output distributions for each hidden unit are generated by a sigmoid function of the weighted sum of inputs for each hidden unit. Because the hidden units output 1 or 0, the set of possible feedback vectors into the Elman network is finite and the transition from one feedback vector to another is stochastic. For this reason we can consider Glickman's stochastic Elman networks to be an FSC algorithm. Performance dropped when non-stochastic RNN hidden units were used.

[1]Lin and Mitchell [1992] does apply RNN methods to the reasonably hard Pole Balancing scenario, but it is not clear that memory is necessary to solve this task. Lin's results also show that using finite histories worked better than RNNs for this scenario.

These results show that FSC methods can perform better than other RNNs and finite history methods, and that FSCs can scale to interesting problems. The results also raise the following question: why is policy evolution immune to the undifferentiated I-state problem seen in policy gradient and value function approaches? One explanation is that evolutionary methods do not get stuck in local maxima. Evolutionary systems keep generating random agents until one works.

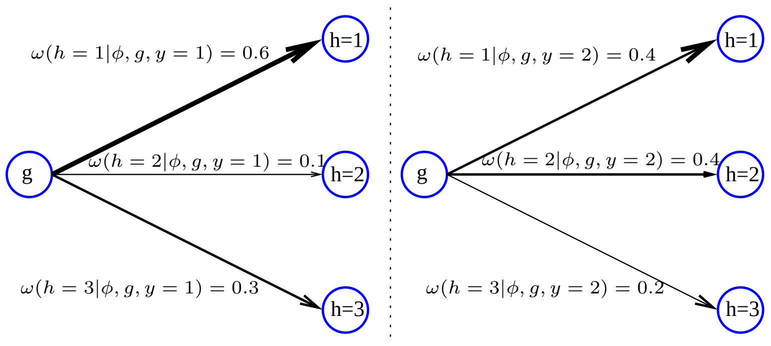
Normally, gradient methods rely on converging to a local maximum from some set of initial parameters that makes as few assumptions as possible about the scenario. In the case of policy-gradient algorithms this is the set of parameters that generates uniform $\omega(\cdot|\phi, g, y)$ and $\mu(\cdot|\theta, g, y)$ distributions. However, Theorem 6 tells us that this initialisation will result in $\nabla^{\phi}\eta = 0$. We tried to apply the trick used in neural networks for similar situations: initialising the weights with small random values. However, to obtain a reasonable gradient we had to start with large weights that begin to saturate the soft-max function. Consequently, the system again starts near a poor maximum.

Thus, we are caught between two competing sets of maxima: (1) FSC transition gradients are small for small weights, (2) FSC transition gradients are small for large weights. Empirically we found the best middle point was to initialise weights randomly between $[-0.5, 0.5]$, however convergence was unreliable. For example, the trivial Load/Unload scenario would fail to converge around 50% of the time. The next section describes a simple trick that allows us to obtain reliable convergence on the Load/Unload scenario and allows us to scale FSC methods to more complex scenarios.

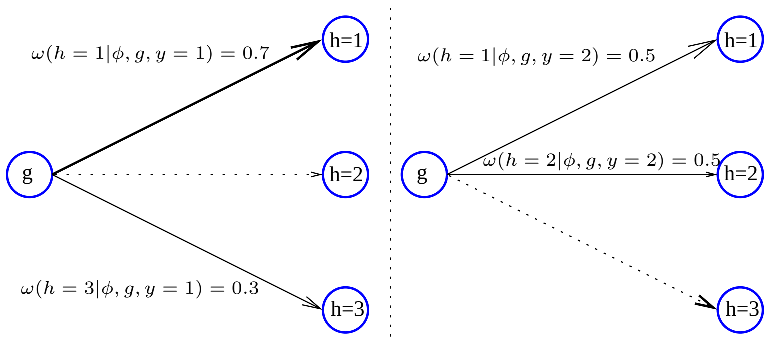
7.2 Sparse FSCs

Theorem 6 tells us what conditions to avoid when choosing initial parameters for our agent. Our proposed initialisation scheme adds necessary structure, and at the same time reduces the computational complexity of each step, by taking advantage of the fact that the imposed structure reduces the number of parameters that need to be estimated.

We propose a modified I-state representation: a large sparse FSC where all I-states have out-degree $k \ll |\mathcal{G}|$. We randomly disable all but k transitions out of each I-state for each observation. We minimise the chance that I-state trajectories overlap for different observation trajectories by ensuring that, for all I-states, no two observations share the same set of outward transitions. Figure 7.1 illustrates a $|\mathcal{G}| = 3$ FSC fragment with transitions from node g to node h. The upper figure shows the fully connected FSC with non-uniform transition probabilities. The lower figure shows the sparse version with $k = 2$. The probabilities and gradients for the disabled transitions are not computed and are effectively 0.


$\omega(h = 1|\phi, g, y = 1) = 0.6$
$\omega(h = 2|\phi, g, y = 1) = 0.1$
$\omega(h = 3|\phi, g, y = 1) = 0.3$
$\omega(h = 1|\phi, g, y = 2) = 0.4$
$\omega(h = 2|\phi, g, y = 2) = 0.4$
$\omega(h = 3|\phi, g, y = 2) = 0.2$
g
h=1
h=2
h=3

(a)


$\omega(h = 1|\phi, g, y = 1) = 0.7$
$\omega(h = 3|\phi, g, y = 1) = 0.3$
$\omega(h = 1|\phi, g, y = 2) = 0.5$
$\omega(h = 2|\phi, g, y = 2) = 0.5$
g
h=1
h=2
h=3

(b)

Figure 7.1: This figure demonstrates a transition from I-state $g \to h$ for an FSC with 3 I-states. Figure 7.1(a) shows the fully connected version. The distribution on the left side is generated by observation $y = 1$; the distribution on the right side is generated by observation $y = 2$. Figure 7.1(b) shows one possible sparse version of this FSC. It is important that the possible transitions for observation $y = 1$ are different to the transitions for $y = 2$.

This trick imposes the necessary constraints on the FSCs without requiring domain knowledge, and without requiring parameters to be initialised to non-zero values. Using sparse FSCs observation/action trajectories will generate minimally overlapping distributions of I-state trajectories. Intuitively, this allows the correlation of high-reward observation/action trajectories to a set of I-state trajectories, resulting in the system automatically maximising the probability that those I-state transitions are followed.

The complexity of each IState-GPOMDP step for a lookup table parameterisation is reduced from $O(|\mathcal{U}| + |\mathcal{G}|)$ to $O(|\mathcal{U}| + k)$, allowing the use of many I-states to offset the loss in FSC flexibility. The complexity reduction comes about because if every I-state has only k out transitions, then only k out of $|\mathcal{G}|$ components of the I-state distributions and gradients need to be computed. Exp-GPOMDP has complexity $O(|\mathcal{G}||\mathcal{U}| + |\mathcal{G}|k)$. The additional complexity factor of $|\mathcal{G}|$ restricts the number of I-states it is practical to use. For both algorithms increasing $|\mathcal{G}|$ results in more parameters when using the lookup table representation, requiring more sample steps during early training to gather sufficient statistical evidence to form accurate gradient estimates.

Setting $k = 1$ is an interesting case that forces an observation/action trajectory to always generate the same I-state trajectory. However, I-state trajectories may not be unique, or may merge, so that the FSC "forgets" previous observations. If we use $k = 1$ we need to apply large $|\mathcal{G}|$ to avoid non-unique and merging I-state trajectories. This approach is valid for finite-horizon tasks and is equivalent to an agent that records its entire history $\bar{y}$.

7.3 Empirical Effects of Sparse FSCs

Theorem 6 shows that the gradient magnitude of the stochastic finite state controller is 0 when the transition probabilities are independent of the internal state. This includes the apparently sensible choice of an initial controller that sets all transitions to equal probability (equivalent to $\theta_c = \phi_l = 0\ \forall k, l$ for common parameterisations). A natural question is what happens to the gradient as we approach the uniform case. The next two sections investigate this question empirically, first by using random weight initialisations in a fully connected FSC, and secondly by increasing the degree k of a sparse FSC.

7.3.1 Random Weight Initialisation

Figure 7.2 shows how the magnitude of the initial gradient estimate can vary with the maximum parameter value. The test was conducted using several variants of IState-GPOMDP and IOHMM-GPOMDP. We used the Load/Unload scenario (see Figure 2.2(a)), with $|\mathcal{G}| = 4$. Lookup tables were used as the parameterisation (see Section 3.4.1). The gradient smoothly degraded as we approached uniform distributions, that is, [0] weights. The curve marked "Dense ω" shows results for the worst case scenario of a fully-connected FSC. The rightmost point on this curve represents $\theta_c = \phi_l = 0\ \forall k, l$. The curve marked "Sparse ω" shows results for an FSC with out degree $k = 2$. The curve marked "Det μ" shows the effect of fixing $\mu(u|\theta, h, y)$ to be a deterministic function of (y, g). For this test μ chooses the `left` action if the I-state is 1 or 3, and the `right` action if the I-state is 2 or 4. This is an alternative way to break the zero gradient conditions but makes stronger assumptions about the best FSC policy. In particular, it assumes that the best policy is deterministic. The IOHMM

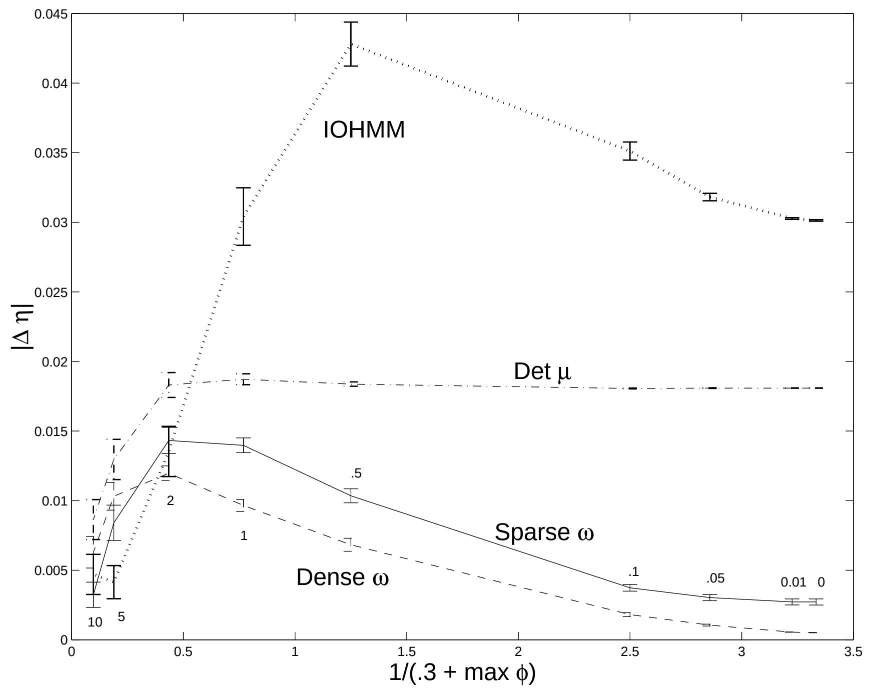


Figure 7.2: Degradation of the magnitude of the gradient for IState-GPOMDP as FSC distributions and action distributions approach uniformity. The lower curves have points labelled with the range of the random initial values for parameters. Results are averaged over 100 runs. Error bars show one standard deviation. The x-axis uses the non-linear scaling $1/(0.3+\max\phi_l)$ to best illustrate the gradient at the extremes of possible parameterisations.

curve shows the magnitude of the θ gradients, computed using IOHMM-GPOMDP after one IOHMM parameter estimation phase.

The results show that moving from a fully connected FSC to a sparse FSC prevents the gradient tending to 0 as the weights approach [0]. It also shows that using a deterministic μ is even more effective, showing no degradation due to small parameter values. The large magnitude of the IOHMM-GPOMDP gradient estimate shows the benefit of computing estimates for smaller set of parameters (the θ parameters only). The dive in gradients for $\max_l \phi_l = \max_c \theta_c > 2$ shows the effect of pushing the parameters into a region where the soft-max function saturates. The peaked shape of these curves illustrates the problem described in Section 7.1, where we are caught between two sets of local maxima.

7.3.2 Changing the Out-Degree

Our definition of sparse FSCs introduces an extra parameter, the out-degree of each I-state k. Figure 7.3 shows how the magnitude of the gradient varied as k was increased for the Load/Unload scenario. Recall that the Load/Unload problem requires only 1 bit of internal state to represent the optimal policy; though in this experiment we chose $|\mathcal{G}| = 10$ to demonstrate the effect of changing k. The top curve represents $\|\nabla\eta\|$ and the lower curve is $\|\Delta_T\|$, estimated using Exp-GPOMDP with an estimation time of $T = 10^5$ steps and $\beta = 0.8$. The initial parameters were set to [0]. At the start of each run a new set of random FSC transitions was selected according to the current value of k. We performed 30 runs for each value of k. As k was increased the gradient magnitude fell until, at $k = 10$, the FSC was dense and $\|\nabla\eta\|$ was within machine tolerance of 0. At that point $\|\Delta_T\| = 3.13 \times 10^{-7}$, which indicates the level of noise in the estimate because the true gradient magnitude is 0. The strongest gradient was at $k = 1$. For infinite-horizon tasks this choice of k is unsuitable because the agent needs to learn useful loops in the FSC, which is impossible if it has no choice of I-state transitions. Thus, $k = 2$ or $k = 3$ seem the most reasonable choices.

Introducing sparsity necessarily restricts the class of agents that can be learnt. The transitions to disable are chosen uniformly at random. This may create an FSC that cannot result in a good policy. It might be useful to generate sparse FSCs using heuristics such as "I-states should always be able to make self-transitions," representing the situation where no additional information needs to be stored at that step. Domain knowledge can also be encoded in the choice of initial FSC.

Section 8.2 demonstrates the effect that using sparse FSCs has on convergence of the Heaven/Hell scenario. Without using sparse FSCs we could not achieve convergence for this POMDP, even using the GAMP algorithm that has access to the model. This evidence, and Theorem 6, shows that the problem is fundamental to the design of the controller and is not merely an artefact of sampling.

7.4 Summary

Key Points

I Initially undifferentiated I-states, or "aliasing on the payoffs," results in failures when trying to learn non-trivial FSCs.

II Initialising the FSC to a large, sparse random structure ensures that the FSC gradient is not initially zero, allowing useful FSCs to be learnt.

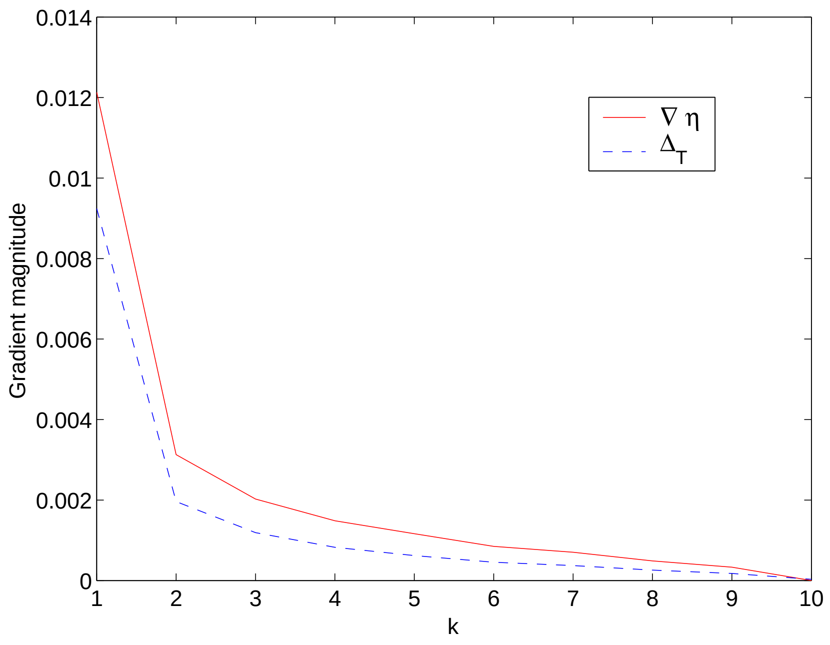


Figure 7.3: Effect of increasing the connectivity for the Load/Unload problem with $|\mathcal{G}| = 10$. Results are averaged over 30 runs.

O┳┳ The gradient drops to zero as the FSC approaches a fully-connected graph with uniform transition probabilities.

Future Work

Making FSCs sparse means the optimal FSC may not be representable. Increasing the FSC size can compensate. In the extreme case of an infinite number of I-states and $k = 1$, we can end up history based methods because the I-state can encode the full history $\bar{y}$. The challenge is to design sparse FSCs for which observation/action trajectories generate minimally overlapping I-state trajectories; and to do this with a finite number of I-states.

It would be interesting to attempt the New York Driving problem to compare our methods to the evolutionary approach of Glickman and Sycara [2001]. The Elman network used has 12 hidden units, thus $|\mathcal{G}| = 2^{12} = 4096$. Our algorithms would not need this many I-states because our policies can make use of the I-states *and observations*. The actions emitted by the Elman network are a function of the I-state only.

Chapter 8

Empirical Comparisons

The meta-Turing test counts a thing as intelligent if it seeks to devise and apply Turing tests to objects of its own creation.

—Lew Mammel, Jr.

This section examines the relative performance of GAMP, IState-GPOMDP, IOHMM-GPOMDP, and Exp-GPOMDP, on three POMDPs from the literature. We begin with the simple Load/Unload scenario, move on to the small but challenging Heaven/Hell scenario and finish with a medium-size robot navigation scenario. All three scenarios are cast as infinite-horizon problems to highlight the ability of the algorithms to work in that context. Within our knowledge, the Heaven/Hell and robot navigation scenarios represent the most difficult POMDPs that have been solved using gradient ascent of FSCs.[1]

Unless otherwise stated, all the agents trained in this section use $\omega(h|\phi, g, y)$ and $\mu(u|\theta, h, y)$ functions parameterised by lookup tables and the soft-max function as described in Section 3.4.1. The table parameters are always initialised to $\phi = \theta = [0]$.

8.1 Load/Unload

Our favourite example, the Load/Unload problem of Peshkin et al. [1999], is described in Figure 2.2(a). The version used in our experiments has 5 locations, the load and unload locations, and 3 intermediate locations that share the same `null` observation. Because the agent can be loaded or unloaded in any of the 5 locations, there are $|\mathcal{S}| = 10$ world states. It is one of the simplest POMDPs that requires memory to solve (provided the agent does not know what its last action was). There is no noise added to the problem and we require $|\mathcal{G}| = 2$, that is, 1-bit, to represent the best finite state controller. This deterministic finite state controller is illustrated by the policy graph in Figure 2.2(b).

[1]All the POMDPs in this thesis, except the speech experiments, are available in Anthony Cassandra's POMDP format from `http://csl.anu.edu.au/~daa/pomdps.html`

8.1.1 Experimental Protocol

With the exception of IOHMM-GPOMDP, all the algorithms in this thesis will reliably solve the simple Load/Unload problem given sufficient resources. For the trajectory sampling algorithms we set the number of gradient estimation steps to be $T = 5000$, deliberately under-estimating the number of steps required to help gauge the relative performance of the algorithms. A discount factor of $\beta = 0.8$ is sufficient for this simple problem. All runs used $|\mathcal{G}| = 4$ for a total of $4 \times 10 = 40$ states. We did not use the minimum possible $|\mathcal{G}| = 2$ because this value is too small to allow a sparse FSC with $k > 1$.

Recall from Section 3.5 that a quadratic penalty on the parameters can be applied to help avoid poor maxima. For the Load/Unload experiments we did not apply penalties because the scenario is trivial enough to allow good solutions without them, and over-penalising can mask the true performance of the algorithms. Quadratic penalties were necessary to get good results for the Heaven/Hell and robot navigation experiments.

For the IOHMM-GPOMDP runs we took samples of 1000 steps in order to re-estimate the IOHMM parameters. We used an IOHMM training algorithm that is a minor modification of the Baum-Welch procedure for standard HMMs. The procedure is described in Appendix D.2 and Algorithm 9. The Load/Unload IOHMM indexes state transitions distributions by the current observation and IOHMM state. In order to achieve convergence the IOHMM had to distinguish between rewards for loading and rewards for unloading. To do this we used a reward of 1.0 for loading and 2.0 for unloading. This allowed us to use discrete emission distributions with 3 symbols: $r_t = 0$, $r_t = 1$ and $r_t = 2$. This does not change the optimal policy and the quoted results have been adjusted back to the normal reward schedule.

The total number of parameters for all but the IOHMM-GPOMDP algorithm was 56 for dense FSCs and 32 for sparse FSCs. IOHMM-GPOMDP cannot use the lookup-table parameterisation for $\mu(u|\theta, h, y)$ because the I-state input h is replaced by α_t, the belief vector for IOHMM-state occupancies. Instead, we parameterised $\mu(u|\theta, \alpha, y)$ using an ANN, depicted in Figure 6.1. The computation of $\nabla\mu(u|\theta, h, y)$ was described in Section 3.4.2. Using a linear ANN (no hidden layer), IOHMM-GPOMDP required $|\mathcal{G}|^2|\mathcal{Y}| = 48$ IOHMM parameters and $|\mathcal{U}|(|\mathcal{G}| + |\mathcal{Y}| + 1) = 16$ ANN parameters, for a total of 64 parameters. The $+1$ term comes from the constant bias input supplied to the ANN.

The Incremental Pruning algorithm [Zhang and Liu, 1996] results are provided to allow comparison with exact methods. We used Anthony Cassandra's `pomdp-solve` V4.0 source code that performs value iteration, learning value functions represented by convex piecewise linear functions as described in Section 2.5.1.2. This code outputs a policy graph that can be loaded by our code to obtain results for comparison.

8.1.2 Results

Table 8.1 summarises the results for this problem on all the algorithms described in this thesis, using both sparse FSCs and dense FSCs. The rightmost column represents the mean time in seconds taken to converge to a value of η that represents a reasonable policy. Only those runs that achieve the cut-off value are included in the convergence time results, but all runs are included in the mean and variance statistics. The number of runs that reach the cut-off is written in brackets next to the convergence time. This allowed a comparison of the running times of different algorithms that may not always achieve the best policy.

Several interesting observations arise from Table 8.1. The first is the complete failure of GAMP to learn when a dense initial FSC is used. This is an expected consequence of Theorem 6 which tells us that the gradient of the ϕ parameters is always 0 in this case. As expected, the dense Exp-GPOMDP algorithm also failed, but both algorithms do well when we use sparse FSCs. The IState-GPOMDP algorithm with a dense FSC occasionally succeeds because noise in the gradient estimates move the parameters so that Theorem 6 no longer applies. This does not often happen for GAMP and Exp-GPOMDP, probably due to their lower variance.

The GAMP algorithm performs consistently well for sparse FSCs and is the fastest algorithm. The GAMP gradient estimates have zero variance, but there is variance in the long-term average reward due to the random choice of sparse FSC. As expected, the sparse version of Exp-GPOMDP performs significantly better than the sparse IState-GPOMDP algorithm.

We could not get the IOHMM-GPOMDP algorithm to converge reliably, and although it outperforms the IState-GPOMDP algorithm in this experiment, simply increasing the gradient estimation time allowed IState-GPOMDP to converge on every run. This is consistent with the problems identified in Section 6.1.4. Figure 8.1 shows how quickly the three simulation based algorithms converge for sparse FSCs. Exp-GPOMDP clearly outperforms the other two. According to a single-tailed t-test the IOHMM-GPOMDP algorithm is statistically better than IState-GPOMDP with a confidence of 95%. The Exp-GPOMDP result is significantly better than IState-GPOMDP with a 99% confidence interval. Although not shown on the graph, the GAMP result is statistically significantly better than Exp-GPOMDP with 99% confidence.

Unsurprisingly, the exact Incremental Pruning method works well on this small problem, always obtaining the optimal policy. Using VAPS in an episodic setting of the Load/Unload problem achieved convergence within around 20,000 steps [Peshkin

Table 8.1: Results over 100 training runs on the Load/Unload scenario with 5 positions. η values are multiplied by 10. The variance has also been scaled to match. Mean η is the mean of all η values achieved over the 100 agents. Max η is the η achieved by the *best* of the 100 agents trained. Five of the seven algorithms (excluding Incremental Pruning) trained at least one agent to optimum performance.

Algorithm	*mean* η	*max.* η	*var.*	*secs to* $\eta = 2.0$	
GAMP dense	0.500	0.500	0	—	(0)
GAMP $k = 2$	2.39	2.50	0.116	0.22	(96)
IState-GPOMDP dense	0.540	2.48	0.0383	2.75	(1)
IState-GPOMDP $k = 2$	1.15	2.50	0.786	2.05	(31)
Exp-GPOMDP dense	0.521	0.571	6.05×10^{-4}	—	(0)
Exp-GPOMDP $k = 2$	2.18	2.50	0.340	9.75	(82)
IOHMM-GPOMDP	1.41	2.50	0.696	5.53	(46)
Inc. Prune.	2.50	2.50	0	3.27	(100)
Optimum	2.50				

et al., 1999]. Their version of the problem provided information about which position the agent was in on the road, rather than the `null` observations we give for non-end points. It is not clear if this would help or hinder the agent in the long run because the extra parameters needed to handle the extra observations may slow learning down. The finite-horizon setting used in that paper makes the problem easier because unbiased Monte-Carlo gradient estimates can be computed. Relatively short episodes can be used to prevent the temporal credit assignment problem becoming too hard, and the variance can kept low by averaging gradients over many episodes to form a final estimate.

8.2 Heaven/Hell

The Heaven-Hell problem of Geffner and Bonet [1998], and Thrun [2000], is shown in Figure 8.2. The agent starts in either position shown with probability 0.5. The location is completely observable except that the agent does not initially know if it is in the left world or the right world. The agent must first visit the signpost that provides this information, and remember the sign direction until the top-middle state. The agent should then move in the direction the signpost points to receive a reward of 1, or -1 for the wrong direction. In theory, 3 I-states are sufficient for optimal control: one for "sign post not visited," one for "sign post points left," and one for "sign post points right."

Although this problem has only 20 world states, it contains two features that make

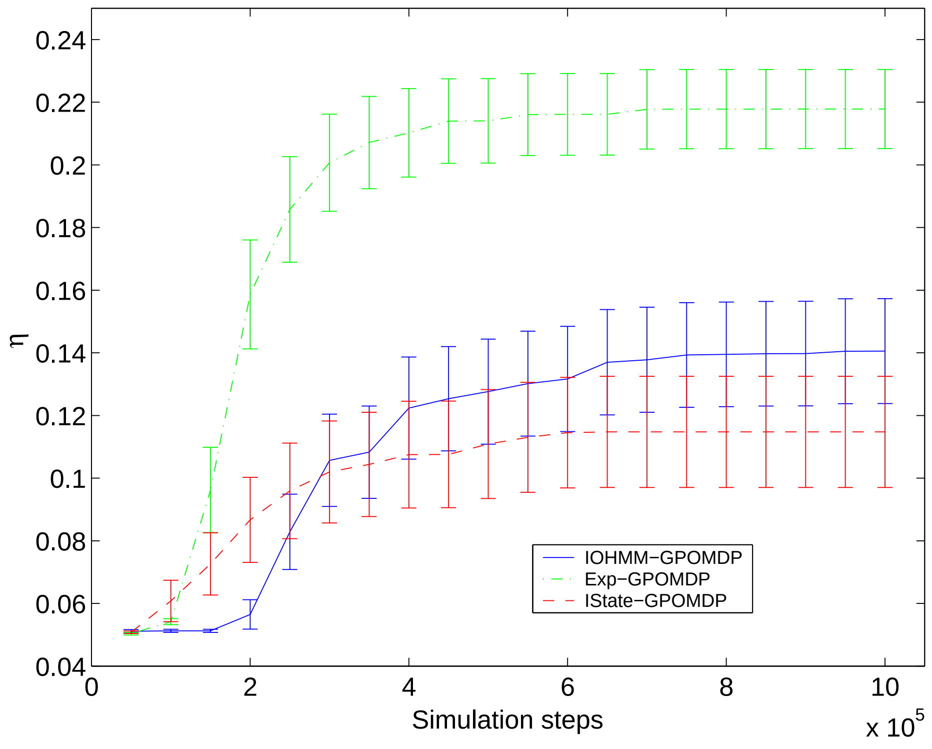


Figure 8.1: Selected convergence plots for the Load/Unload scenario, comparing 100 runs of IState-GPOMDP and Exp-GPOMDP. We used 4 I-states and a connectivity of $k = 2$. This figure also shows the results for IOHMM-GPOMDP trained using an IOHMM with 4 states and 1000 observations per re-estimation. The variance bars show 0.2×standard deviation to reduce clutter.

it a difficult task. Firstly, it requires long-term memory. The signpost direction must be remembered for 5 time steps. Secondly, the temporal credit assignment problem is hard because the actions critical to receiving a consistent reward happen up to 11 steps before receiving the reward. There is nothing in the immediate reward structure that informs the agent that visiting the signpost is good, that is, for a random policy the same average reward of 0 is received regardless of whether the agent saw the signpost.

8.2.1 Experimental Protocol

The global state space comprised of 20 world states and 20 I-states, for a total of $|\mathcal{S}||\mathcal{G}| = 400$. We used more I-states than necessary to increase the ease with which the sparse FSC learnt cycles in the policy-graph of the appropriate length. An analogy can be drawn with neural networks where convergence can be hastened by using more hidden units than is strictly necessary.

IState-GPOMDP used a discount factor of $\beta = 0.99$, and gradient estimation times

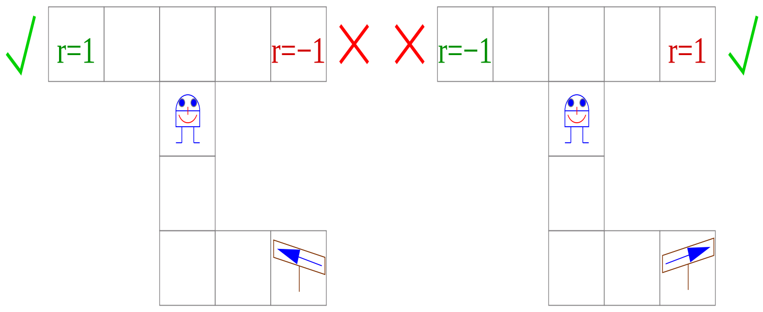


Figure 8.2: The Heaven/Hell problem: the optimal policy is to visit the signpost to find out whether to move left or right at the top-middle state in order to reach heaven.

of 10^7 steps. To choose β we incorporated a discount factor into GAMP[2] and experimented with different values of β. Discounted GAMP produces gradient estimates that are equivalent to IState-GPOMDP or Exp-GPOMDP estimates with infinite estimation times. As $\beta \to 1$ the estimate converges the true gradient. We reached the conclusion that $\beta = 0.99$ is roughly the smallest discount factor that still allowed reliable convergence. Conjugate-gradient convergence took more than twice as long using GAMPwith $\beta = 0.99$ than it took using un-discounted GAMP. The long-term reward η often reached a plateau and took a long time to start improving again. The smallest feasible discount factor is important because the variance of the gradient estimates increases with β, as described in Section 5.2. We discuss discounted GAMP further in Section 8.2.3.

The lookup table for $\omega(h|\phi, g, y)$ and $\mu(u|\theta, h, y)$ contained a total of 1540 parameters when the out degree of the I-states was $k = 3$. Quadratic penalties were used to avoid local maxima. GAMP used a penalty of $\wp = 1 \times 10^{-7}$. Good convergence was found to be sensitive to the penalty, with more runs failing when the penalty was set to 1×10^{-6} or 1×10^{-8}. IState-GPOMDP was less sensitive to the penalty, which we set to $\wp = 1 \times 10^{-4}$.

We ran Incremental Pruning for this problem, however it failed to converge after running for 100,000 seconds, producing policy graphs with $|\mathcal{G}| > 2000$ states when the optimal infinite-horizon policy graph can be represented with $|\mathcal{G}| = 8$.[3] This poor performance resulted from the failure of Incremental Pruning to identify that the task was actually episodic and that the belief state should be reset after receiving a reward. Using a fixed number of I-states forces our algorithms to identify useful cycles in the

[2]A discount factor can be added to GAMP by replacing Line 11 of Algorithm 1 with $v = \beta P v$.

[3]Exact algorithms compute policy graphs equivalent to a deterministic policy, only allowing one action per node. Because we make the policy $\mu(u|\theta, h, y)$ a function of current observation as well as the I-state (equivalent to a policy-graph node), we can compute optimal policies with far fewer I-states. In the case of Heaven/Hell, $|\mathcal{G}| = 3$ is sufficient.

Table 8.2: Results over 10 runs for the Heaven/Hell scenario. Values for η have been multiplied by 10^2.

Algorithm	*mean* η	*max.* η	*var.*	*secs to* $\eta = 5.0$	
GAMP $k = 3$	9.01	9.09	0.0514	34	(10)
GAMP dense	5.24×10^{-3}	5.24×10^{-3}	0	—	(0)
IState-GPOMDP $k = 3$	6.49	9.09	10.8	11436	(8)
IState-GPOMDP dense	0.0178	0.0339	3.23×10^{-5}	—	(0)
Optimum	9.09				

policy graph, hence identifying the episodic qualities of the POMDP such as the 11 step cycle of the optimal agent.

8.2.2 Results

The experiments were run with dense initial FSCs (5280 parameters) and with sparse random FSCs of out degree $k = 3$ (1540 parameters). The stochasticity in the results reported for the GAMP algorithm comes from the random choice of initial FSC.

Table 8.2 shows the results of these experiments. Both GAMP and IState-GPOMDP found the optimal agent, but only GAMP found it consistently. The 2 failed IState-GPOMDP runs had $\eta \approx 0$. All GAMP runs with $k = 3$ succeeded. All the runs with dense FSCs failed. For a dense FSC, the initial gradient magnitude estimated by GAMP is within machine tolerance of 0. In successful runs the agents typically learnt to use about half of the available I-states, never making transitions to the unused-states.

The wall clock times to convergence quoted for these experiments are not directly comparable. The IState-GPOMDP experiments were run using 94 processors of a 550 MHz dual CPU PIII Beowulf cluster. The GAMP experiments were run on a 1.3 GHz Athlon, roughly equivalent to 3 CPUs of the cluster. It is extraordinary that GAMP converged in a small fraction of the time required by IState-GPOMDP, with better results. This demonstrates the advantage of having a model of the world.

We verified that Exp-GPOMDP can learn to visit the signpost. However, since 1 run takes more than 2 days on our cluster, we were prohibited from a formal comparison. Both IState-GPOMDP and Exp-GPOMDP tended to spend about half the total number of estimation steps stuck near their starting point before locking onto a possible solution and improving rapidly.

The same scenario was tackled by Geffner and Bonet [1998] and a continuous state space version was tackled by Thrun [2000]. Both of those papers assumed knowledge of the POMDP model and that the scenario is episodic. Within our knowledge this is the first time the Heaven/Hell scenario has been solved using a model-free algorithm.

We could not get IOHMM-GPOMDP to solve Heaven/Hell.

8.2.3 Discounted GAMP

Using the GAMP estimator to choose an appropriate value of β for our IState-GPOMDP experiments is worth extra discussion. Choosing β using discounted GAMP is not generally feasible unless we could have trained the agent using GAMP in the first place. However, it is instructive to explore the gradient ascent performance of discounted GAMP. A GAMP estimate with discount β is equivalent to an IState-GPOMDP or Exp-GPOMDP estimate with the same β and an infinite number of steps T. This eliminates one source of uncertainty about our estimates. For example, we can compare the GAMP gradient estimates for $\beta = 1.0$ and $\beta = 0.99$ to determine the *true* bias induced by the discount factor. We no longer have the added uncertainty of noise in the estimates.

Baxter and Bartlett [2001] show that $1/(1-\beta)$ must be greater than the mixing time τ to achieve good estimates. Therefore, discounted GAMP could be used as an empirical tool to estimate the mixing time τ of the POMDP. By plotting the angle between GAMP estimates with $\beta = 1$ and $\beta < 1$, we can estimate τ by looking at the value of β where estimates start becoming "close" to the true gradient. However, the mixing time changes as the parameters converge. We could plot the error in the estimate for one or more values of β over the duration of training to gather an insight into how τ varies as the agent improves.

The poor performance of GAMP when $\beta < 0.99$ indicates that the mixing time of the Heaven/Hell scenario is at least hundreds of steps. Intuitively, this means it takes hundreds of steps for the effects of previous actions to stop having a significant impact on the current state. This fits well with our understanding of why the scenario is hard. For example, rewards are delayed by a *minimum* of 11 steps from the relevant actions.

We also mentioned that the value of η tends to plateau when GAMP with $\beta = 0.99$ is used during gradient ascent. We expect that these plateau regions correspond to parameter settings that demonstrate mixing times even larger than a few hundred steps. Figure 8.3 shows that this is the case. We performed one conjugate-gradient ascent training run of the Heaven/Hell scenario using GAMP with a discount factor of $\beta = 0.99$. All the other parameters were the same as those used to generate the GAMP results reported in Table 8.2. At the start of training the discount/bias curve indicates that $\beta = 0.99$ induces an error of only 0.42° in the gradient estimate.[4] After training for a few seconds on a single processor machine, the long-term average reward increased from the start value of 0 (within machine precision) to 2.74×10^{-6}. At this point training reached a plateau and we re-calculated the discount/bias curve. The

[4]Additionally, at the start of training undiscounted GAMP has an error of 0.0003° compared to the true gradient computed by evaluating Equation (4.6) directly.

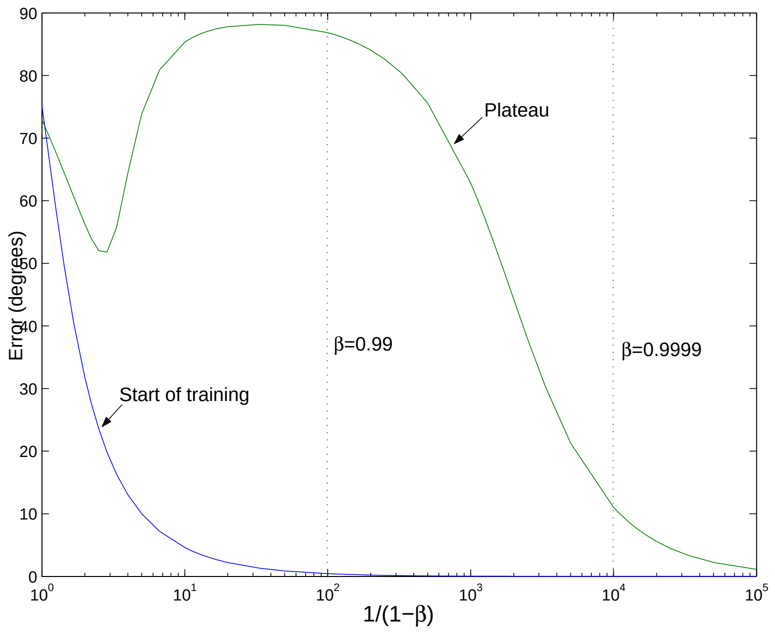


Figure 8.3: An illustration of how the discount factor introduces bias into the GAMP estimates for the Heaven/Hell scenario. The lower curve represents the discount/bias curve at the start of training. The upper curve represents the discount/bias curve when training was observed to have reached a plateau in performance. The x-axis can be thought of as the proposed mixing time τ, modulo constants and log factors. Training was performed using $\beta = 0.99$. This is sufficient at the start of training, but results in a large bias in the region of parameter space corresponding to the upper curve.

bias for a $\beta = 0.99$ is now 86.8°. To bring the bias under 1° requires $\beta > 0.99999$. Given this result, it is impressive that IState-GPOMDP found the optimal policy with $\beta = 0.99$. The local minimum in the plateau curve around $\beta = 0.65$ is an interesting feature. This may indicate that if short term rewards are allowed to dominate (using $\beta = 0.65$) then the gradient is "somewhat right," and considering medium term rewards just confuses the estimate; but to get the true gradient, long term rewards must be considered. The bias is lower at the start of training because the gradient is dominated by "easy" to learn concepts.[5] Once the easy concepts are learnt, the long mixing time of the scenario becomes apparent. When $\beta = 1.0$, conjugate-gradient ascent led quickly to the optimal agent. Because we used a particular random FSC, this is an illustrative

[5]We analysed the agent's behaviour after it reached the plateau. It had learnt a slight preference for moving left or right at the intersection, depending on which I-state it is was in. The FSC had not evolved any discernable structure. We interpret this as meaning there was some initial correlation between the I-states and which way the sign-post pointed (if it was visited), but the gradient signal for the ϕ parameters was very weak. The FSC transitions that would increase the correlations were not being reinforced.

example rather than a representative experiment.

Because the variance of IState-GPOMDP estimates scales with $1/\left(T(1-\beta)\right)$, a good knowledge of how β should be set allows the variance to be controlled by selection of an appropriate estimation time T.

8.3 Pentagon

An interesting problem domain is defined by Cassandra [1998]. In these scenarios a robot must navigate through corridors to reach a goal. The world is mapped into discrete locations. The robot occupies a location and points North, South, East or West. The actions it can take are $\mathcal{U} = \{$`move forward`, `turn left`, `turn right`, `declare goal`$\}$. Actions do nothing with 11% probability, or move/turn one too many steps with 1% probability. This models the unreliability of the movement systems. There are 28 observations that indicate whether the locations immediately to the front and sides of the robot are reachable. The observations have a 12% chance of being misleading, modelling sensor uncertainty. We modified the problem slightly from the original definition to make it an infinite-horizon task and to make rewards control-independent. The behaviour is changed so that a `declare goal` action in the goal location causes a transition to an added state where a reward of 1 is received prior to the agent being taken back to the start state. No penalty is received, and the agent is not returned to the start state, for the `declare goal` action in the wrong state. There is no cost associated with movement because to obtain the highest average reward in the infinite-horizon setting the agent must minimise the number of movements. In this experiment the agent always starts in the same place but the noise means the agent can quickly become confused about its location even if it has a perfect memory of past observations and actions.

Figure 8.4 shows the Pentagon navigation scenario where the robot must move from the bottom left state to the walled state in the middle region. This problem exhibits a lot of symmetry, meaning the agent finds many states hard to distinguish by observations alone.

8.3.1 Experimental Protocol

This scenario has 52 locations, 4 directions plus one "goal obtained" state, for a total of 209 states. Using 20 I-states the global-state space size is $20 \times 209 = 4180$ states.

The IState-GPOMDP algorithm required a gradient estimation time of 2×10^6 steps to achieve gradient estimates that were consistently within $90°$ of each other. This scenario required estimating up to 3920 parameters. Random directions sampled from such high-dimension parameter spaces tend to be orthogonal with probability close to

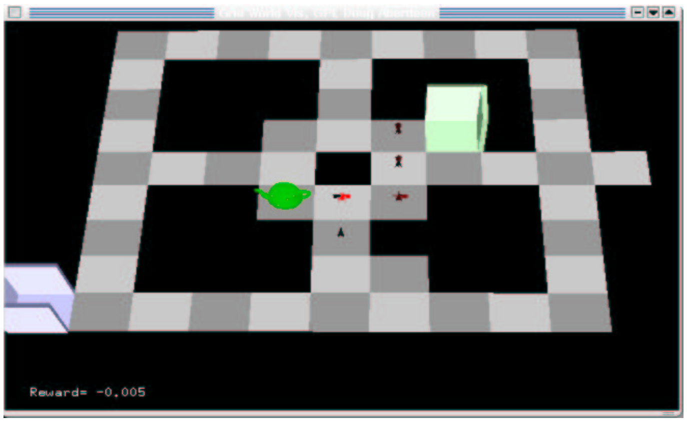


Figure 8.4: The Pentagon robot navigation problem maze.

1. Thus, any degree of consistency in the gradient estimates indicates a gradient signal is being extracted from the noise. Exp-GPOMDP required 10^6 steps, demonstrating an effective reduction in variance. Each point in the gradient-ascent line search used an estimation time of one quarter of the standard estimation time. This speeds up convergence, taking advantage of the GSEARCH algorithm's robustness against variance during the line search (see Appendix B.1). For the Monte-Carlo algorithms a quadratic penalty of $\wp = 10^{-4}$ was used. A discount factor of $\beta = 0.99$ was chosen to match that used by Cassandra [1998]. GAMP used a penalty of $\wp = 10^{-5}$.

The experiment labelled Belief in Table 8.3 used a different algorithm to those described so far. Belief uses the POMDP model and the observations to maintain a belief state over the 209 world states. Recalling Section 2.5.1.1, we denote the world-state belief with b_t. The belief state b_t is passed to IState-GPOMDP in place of the observations y_t. As the belief state is a sufficient statistic for all past events [Åström, 1965], the optimal policy needs no memory, and so we set the number of I-states to 1. Because belief states are vectors in $[0, 1]^{|\mathcal{S}|}$, we cannot use tables to parameterise policies. Instead we use a linear controller policy with 209 inputs and 4 outputs passed through a soft-max function to generate action distributions, as described in Section 3.4.2.

This is similar to the IOHMM-GPOMDP algorithm with one major and one minor difference. The major difference is that instead of learning the parameters of an IOHMM, the POMDP model provides the transition and emission probabilities of an $|\mathcal{S}|$ state IOHMM. The $q(j|i, u)$'s give the transition probabilities, and the $\nu(y|i)$'s give the emission probabilities. The world-state belief update is given by Equation (2.7). The minor difference is that the ANN no longer needs to receive the observation y_t because it is sufficiently summarised by the belief state. The Belief experiment uses a deterministic belief state update $\omega(b_{t+1}|\phi, b_t, y_t) = 1$, and the stochastic policy $\mu(u_t|\theta, b_{t+1})$.

Table 8.3: Results over 10 runs for various algorithms on the Pentagon scenario. Values for η have been multiplied by 10^2.

Algorithm	$\lvert\mathcal{G}\rvert$	k	*mean* η	*max.* η	*var.*	*secs to* $\eta = 2.0$	
GAMP	5	2	2.55	2.70	0.0102	611	(10)
	10	2	2.50	2.63	0.0128	4367	(10)
	20	2	2.50	2.80	0.0250	31311	(10)
	20	3	2.89	3.00	0.00613	47206	(10)
IState-GPOMDP	5	2	2.06	2.42	0.281	649	(9)
	10	2	2.18	2.37	0.0180	869	(9)
	20	2	2.12	2.28	0.0137	1390	(9)
	20	3	2.15	2.33	0.0138	1624	(9)
Exp-GPOMDP	5	2	1.96	2.33	0.401	1708	(7)
	10	2	2.19	2.40	0.0151	6020	(9)
	20	2	2.11	2.27	0.0105	29640	(8)
	20	3	2.26	2.36	0.00448	48296	(10)
IOHMM-GPOMDP	1		1.26	1.39	0.130	—	(0)
IOHMM-GPOMDP	5		1.47	1.63	0.365	—	(0)
IState-GPOMDP	1	1	1.35	1.37	0.000324	—	(0)
Belief			2.67	3.65	0.778	2313	(7)
MDP			4.93	5.01	0.00148	24	(10)
Noiseless			5.56	5.56	NA		

As in the Load/Unload experiment, the IOHMM experiment used discrete transition distributions indexed by IOHMM-state and observation y_t. Unlike the Load/Unload experiment, reward emissions were generated by a single Gaussian for each state, bounded to $[-1, 1]$. Because a Gaussian is described by two parameters, the mean and variance, we achieve a more compact model of the rewards when the POMDP can emit a large or continuous range of rewards. In the case of our definition of the Pentagon problem, there is no compelling reason to prefer the Gaussian representation because only rewards of 1 or 0 are issued. We used it to demonstrate the alternative to the discrete distributions used for the Load/Unload experiment. Training Gaussian emission distributions for IOHMMs is no different to training them for standard HMMs. The details are provided in Appendix D.2 and Algorithm 9 was used for IOHMM training.[6]

8.3.2 Results

The lower part of Table 8.3 provides baseline results for comparisons. Unless we can guarantee that we have provided enough I-states to represent the optimal policy, using

[6]Algorithm 9 is the IOHMM training procedure for discrete distributions. The only modification needed for the Gaussian reward distributions is to replace Line 20 with Equations (D.5) and (D.6).

an FSC will only provide an approximation to the optimal agent. Recall that the optimal agent is defined as best agent we could train assuming access to the full belief state. The Belief algorithm uses the full belief state, thus the Belief result of $\eta = 3.65$ (from Table 8.3) is a rough empirical *upper* bound for the η we should be able to achieve using a large FSC. On the other hand, IState-GPOMDP with $|\mathcal{G}| = 1$ is a memory-less policy that provides a rough empirical *lower* bound of 1.37 for the results we expect to achieve using FSCs.

The long-term average rewards that are between these empirical bounds for $|\mathcal{G}| > 1$ show that FSCs can be used to learn better than memory-less policies by using a *finite amount of memory.* The MDP row gives the results for learning when the agent is told the true state of the world, giving us an empirical value for the best policy achievable in a fully observable environment. The Noiseless row is the theoretical maximum MDP reward that can be achieved if no observation or transition noise is present.

The Incremental Pruning algorithm aborted after 5 value-iteration steps, consuming all 256 MB of memory on an Athlon 1.3 GHz machine in 10,800 seconds. Cassandra [1998] contains results for the Pentagon problem that were obtained using the most likely state heuristic (see Section 2.5.2.1). This method uses the belief state only to identify the most likely current state, then performs value updates based on the assumption that the system is in that state. This heuristic greatly simplifies the problem of representing the value function. To perform a comparison we ran our trained policies on the original POMDP and used the sum of discounted rewards criterion as specified in Cassandra's thesis. The maximum Belief result of 3.65 gives a discounted reward of 0.764. This result sits between Cassandra's results of 0.791 for a known start state and 0.729 for a uniform initial belief state. We do not reset the I-state after receiving rewards so the state in the initial position is not fixed, however the probability of occupying each I-state is not uniform, thus Cassandra's results bracketing ours is a reasonable outcome.

Because the Belief algorithm had access to the world-state belief (using the model), it is unsurprising that it obtained the best maximum η. GAMP also used a model but restricted memory to $|\mathcal{G}|$ states, resulting in the second highest maximum. GAMP may sometimes be preferable to Belief because of its zero-variance (the variance in the tables is due to the random choice of sparse FSC), supported by the fact that for $|\mathcal{G}| = 20$ the GAMP *mean* is better than Belief's (but not with more than 90% t-test confidence).

Increasing $|\mathcal{G}|$ and the FSC connectivity k generally improves the results due to the increasing richness of the parameterisation, however, IState-GPOMDP performed best for $|\mathcal{G}| < 20$ because we did not scale the number of gradient estimation steps with $|\mathcal{G}|$. Exp-GPOMDPs reduced variance allowed it to improve consistently as $|\mathcal{G}|$ was increased, while still using fewer estimation steps than IState-GPOMDP. Figure 8.5 shows that for

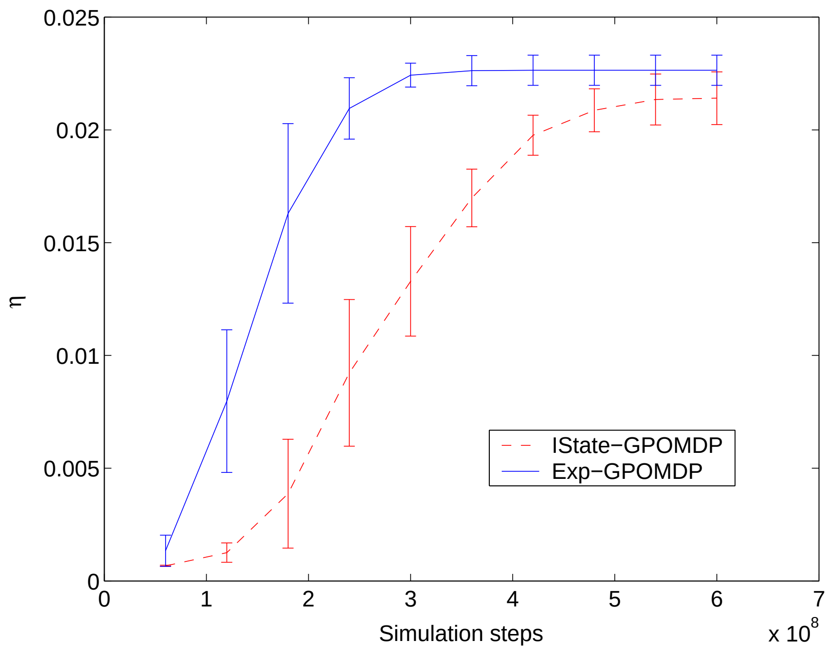


Figure 8.5: Convergence of IState-GPOMDP and Exp-GPOMDP on the Pentagon problem with $|\mathcal{G}| = 20$ and $k = 3$. Results are averaged over 10 runs.

$|\mathcal{G}| = 20$, Exp-GPOMDP produced a superior result in fewer steps than IState-GPOMDP. The mean result of Exp-GPOMDP is better than IState-GPOMDP with greater than 95% confidence. Setting $k > 3$ caused failures due to small gradients.

8.4 Summary

Key Points

- I We can use FSCs to learn non-trivial tasks with long-term memory and hard temporal credit assignment.
- II The sparse FSC trick works, allowing agents to consistently use multiple bits of memory to their advantage.
- III GAMP performs better than our other algorithms and requires a fraction of the time, demonstrating the advantage of having a model of the world.
- IV Exp-GPOMDP learns with fewer world interactions than IState-GPOMDP but requires roughly $|\mathcal{G}|$ times longer to process each observation. This processing time will often be negligible compared to the time required to carry out an action and get the next observation from the real world.

The IOHMM-GPOMDP algorithm works for the toy Load/Unload scenario but we could not achieve competitive convergence in more complex domains.

Future Work

Much more experimentation needs to be done to determine the effectiveness of FSC methods over a broad range of domains. In particular, the following questions need further investigation:

- How does η scale with the number of I-states for POMDPs that benefit from a large amount of memory?
- Are there methods for generating sparse FSCs that improve performance without increasing $|\mathcal{G}|$?
- How much lower is Exp-GPOMDPs variance compared to IState-GPOMDPs?
- How much harder are these problems made by putting them in infinite-horizon settings?

We have done some brief comparisons with the Incremental Pruning exact algorithm. More appropriate comparisons might be with algorithms such as UTREE (Section 2.6.2.2) and evolutionary training of stochastic recurrent neural networks (Section 7.1). In particular, both of these methods have been applied to the New York Driving scenario of McCallum [1996], providing an obvious choice for future empirical investigation of our Algorithms.

The Belief algorithm was introduced to provide a basis for comparison between our finite-memory algorithms and an algorithm that tracks the full belief state. Belief is another form of model-based policy-gradient estimator. Unlike GAMP, it is still a Monte-Carlo method. Because it is trivial to incorporate belief state factorisation — by providing the state-variable beliefs to the ANN inputs directly — Belief has the potential to work well for large problems without enumerating the state space. This idea was applied to training an ANN to output belief-state *values* [Rodríguez et al., 2000].

If we could extend GAMP to work with continuous observation spaces $\mathcal{Y}$, we could use factored belief states as ANN inputs instead of a finite number of observations. This would provide a relatively fast, zero-variance, low-bias gradient estimator for factored belief-state policy search.

The reader is hopefully convinced that the high variance of the Monte-Carlo gradient estimates can lead to slow convergence. Chapter 9 is devoted to some methods for reducing the variance of the gradient estimates.

Chapter 9

Variance Reduction

[The] error surface often looks like a taco shell.
—Gunnar Rätsch

The results of the previous chapter demonstrate the problems that arise due to the high variance of gradient estimates, that is, we require powerful computers and lots of time to solve problems of any interest. This chapter discusses ways to reduce the variance of these estimates, starting with two novel methods and concluding with some remarks on existing methods.

The first method describes how to add domain knowledge in the form of an infinite impulse response (IIR) filter that replaces eligibility traces. The second briefly outlines how to combine low-variance value function methods to learn the policy, while IState-GPOMDP or Exp-GPOMDP are used to learn the FSC. Then importance sampling is discussed and we show that its application to infinite-horizon problems does not follow immediately from its application to finite-horizon algorithms such as Williams' REINFORCE. Finally, we show that a common variance reduction trick for value-based Monte-Carlo methods can be applied to policy-gradient methods. However, it can cause a form of over-fitting for both value-methods and policy-gradient methods.

9.1 Eligibility Trace Filtering

Lines 4 and 5 of the IState-GPOMDP (see Algorithm 2, Section 5.2) show how to update the eligibility trace z at each time step. This update can be viewed as a first order infinite impulse response (IIR) filter that has an exponential response to an impulse input, plotted in Figure 9.1 [Baxter and Bartlett, 2001]. Impulse response plots, such as Figure 9.1, show the filter output after a filter input of 1 at $t = 0$ and 0 for all $t > 0$. For the remainder of the thesis τ denotes the delay between the action at time $t = 0$ and future rewards that are reaped as a consequence of that action.

The eligibility trace acts as memory, telling the agent what proportion of the current reward each previous action should be credited with. Implicit in this trace update is the assumption that an action is considered to have exponentially less impact on the immediate reward the longer ago the action occurred. This assumption is often

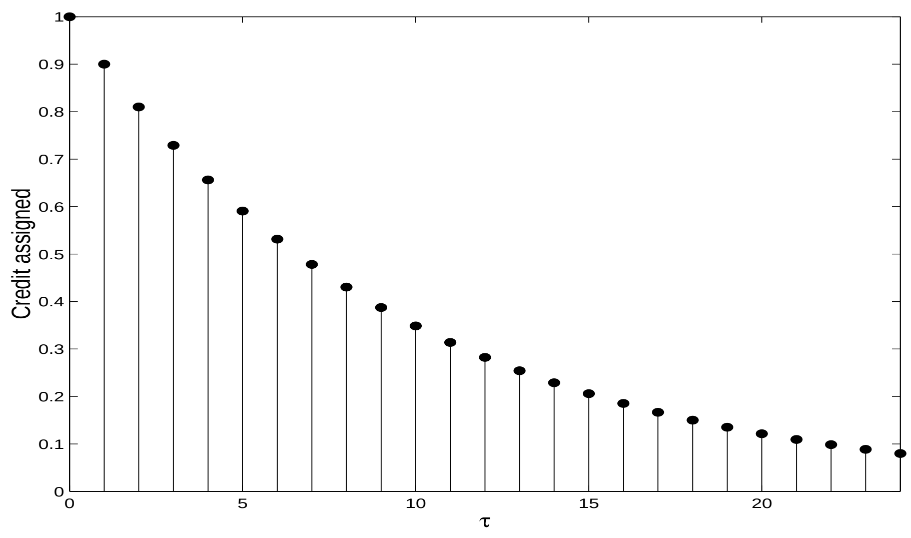


Figure 9.1: The first 25 points of the infinite impulse response generated by the exponential discounting credit assignment model with $\beta = 0.9$. Exponential impulse responses are generated by using the filter coefficients $a = [1.0, -\beta]$ and $b = [1.0]$ in Equation (9.1).

violated, as the experiments of this section will demonstrate. In this section we treat the eligibility trace as an arbitrary infinite impulse response filter, thereby allowing domain specific knowledge about credit assignment to be used.

9.1.1 Arbitrary IIR Trace Filters

Exponential discounting of rewards to actions has nice analysis properties, however in many RL applications we have prior knowledge about the delay between the execution of an action u_t and when rewards $r_{t+\tau}$ are affected by that action. An example is a chemical factory. It takes some time for any new catalyst introduced to have an impact on the yield, thus the temporal credit assignment should ignore actions taken in the intervening time.

By using higher order filters in *any* reinforcement learning algorithm that uses eligibility traces, we can encode prior knowledge about the correct temporal credit assignment. The result is a reduction in the variance of the gradient (or value) estimates. This happens because irrelevant and confusing actions are not considered as candidates to be credited for generating rewards. In effect, we hope to subtract a zero-mean, non-zero variance process from the gradient estimate. IIR filters are a natural choice because the translation from domain knowledge to a good filter is intuitive and extensive literature exists on the design of IIR filters, for example, Elliott [2001]. IIR filters have long been popular for digital signal processing problems such as transmission line equalisation and noise filtering.

Consider the eligibility trace update in line 7 of Algorithm 2. If we let $x_t = \frac{\nabla\omega(g_{t+1}|\phi,g_t,y_t)}{\omega(g_{t+1}|\phi,g_t,y_t)}$ the general IIR form of line 7 is

$$z_{t+1} = -\left(\sum_{n=1}^{|a|-1} a_i z_{t+1-i}\right) + \left(\sum_{n=0}^{|b|-1} b_i x_{t-i}\right), \tag{9.1}$$

where $a = [a_0, a_1, \ldots, a_{|a|-1}]$ and $b = [b_0, b_1, \ldots, b_{|b|-1}]$ are vectors of filter taps with $|a|-1$ being the number of past z's that must be stored and $|b|$ is how many x's must be stored. The tap a_0 is assumed to be 1, which can be ensured by appropriate normalisation. Line 8 of Algorithm 2 can be altered similarly, possibly using a different filter. The extension to any other eligibility trace algorithm, such as SARSA(λ) (see Appendix C.1.1), or Williams' REINFORCE (see Section 2.6.3), is similarly straightforward. We denote the IIR filtered version of Algorithm 2 as IState-GPOMDP-IIR.

9.1.2 Convergence with IIR Trace Filters

The IState-GPOMDP convergence guarantees given by Theorems 4 and 5 in Section 5.2, do not hold for IIR filters because the proofs use the Bellman equation to compute the unknown reward vector r. In this new setting we replace J_β with a filtered version of the rewards J_f so the Bellman equation is no longer directly applicable. Fortunately, we can show that the new z_{t+1} update still estimates an approximation of the gradient that converges to $\nabla\eta$ as the filter impulse response approaches a step function. This result can be used as an alternative proof of Theorem 5:

$$\lim_{\beta\to 1} \pi(\nabla P)J_\beta = \nabla\eta,$$

because setting $\beta = 1$ defines a first order IIR filter with a unit step function response. Without loss of generality the new proofs assume an FIR filter with a possibly infinite number of taps, which includes all possible IIR filters.[1]

Theorem 7. *Define the filtered reward as*

$$J_f(\phi,\theta,i,g) := \mathbb{E}_{\phi,\theta}\left[\sum_{n=0}^{|b|-1} b_n r(i_n,g_n)|i_0 = i, g_0 = g\right],$$

where the expectation is over all world/I-state trajectories. Let $\Delta_T := \left[\Delta_T^\phi, \Delta_T^\theta\right]$ *be*

[1]IIR filters allow a compact representation and implementation of FIR filters with infinitely many taps, but do not enlarge the space of possible filters. Hence, without loss of generalisation, we restrict the proof to FIR filters.

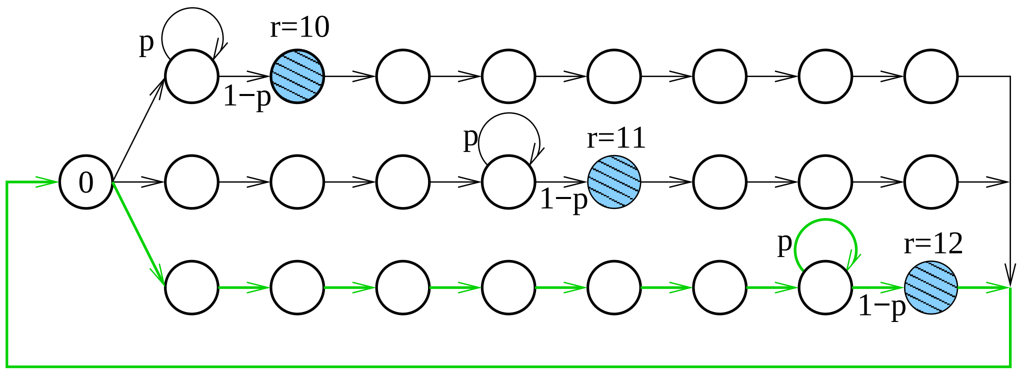


Figure 9.2: The completely unobservable MDP used to test IIR trace filtering in Tests I and II. Test I sets $p = 0.0$; Test II sets $p = 0.5$ so that rewards may occur an arbitrarily long time after the action at state 0. The optimal policy follows the green (thick) transitions.

the estimate produced by IState-GPOMDP-IIR after T iterations. Then under Assumptions 1–4 defined in Section 3.2, $\lim_{T\to\infty} \Delta_T = \pi'(\nabla P)J_f$ *with probability 1.*

This is proved in Appendix A.4.2. The next theorem establishes that if $|b| \to \infty$ and $b_n = 1\ \forall n = 0, 1, \dots, \infty$ — an infinite step response filter — then the approximation $\pi'(\nabla P)J_f$ is in fact $\nabla\eta$.

Theorem 8. *For P parameterised by FSC parameters ϕ and policy parameters θ, then if $b_n = 1\ \forall n = 0, \dots, |b| - 1$, then*

$$\lim_{|b|\to\infty} \pi'(\nabla P)J_f = \nabla\eta.$$

This theorem is proved in Appendix A.4.1. The condition that $b_n = 1$ is not necessary to compute the correct gradient direction. It is sufficient that $b_n = \kappa\ \forall n$, in which case the gradient is scaled by κ.

9.1.3 Preliminary Experiments

We contrived 4 simple POMDPs as test cases:

I. the POMDP described by Figure 9.2 with $p = 0$;

II. the POMDP described by Figure 9.2 with $p = 0.5$;

III. the POMDP described by Figure 9.3 when completely observable;

IV. the POMDP described by Figure 9.3 when only the tree depth is observable.

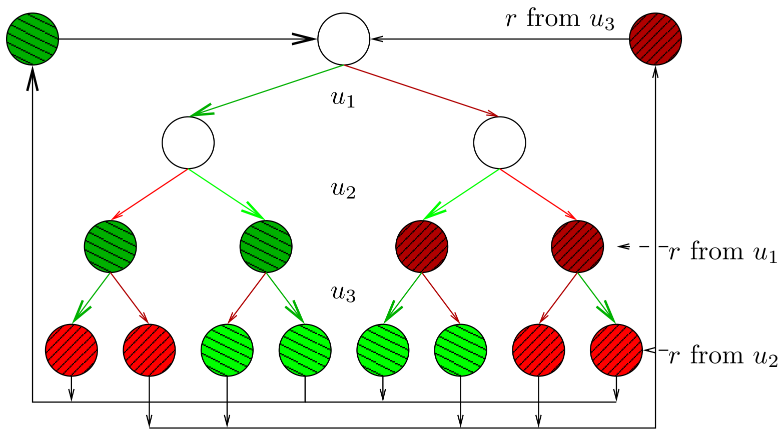


Figure 9.3: The MDP/POMDP used for Tests III and IV. Rewards are issued with a delay of 1 step, in which time a reward can be received for an earlier action. In Test III all states are completely observable, in Test IV, only the depth down the tree is observed. The two nodes that enter the root of the tree are assigned the same depth as the bottom of the tree. Green (light) nodes indicate a reward of 1 and red (dark) nodes indicate a -1 reward.

Tests I and II are examples in which IState-GPOMDP performs poorly but trace filtering performs well. They are based on the completely unobservable MDP shown in Figure 9.2, for which the optimal policy is to follow the lower path. There are 3 actions, all of which only have an effect in state 0, deterministically selecting one of the 3 transitions out of state 0. This POMDP is harder than it looks. Unless $\beta > 0.97$, the reward discounted back to the action at state 0 appears higher for the upper two paths than for the lower optimal path. Thus for $\beta \leq 0.97$, the gradient will point away from the optimal policy.

Tests III and IV are described by Figure 9.3. The agent must fall down the correct branch of the tree to maximise its reward. The reward is always delayed by 1 step, that is, when the agent makes a decision leaving the top node, level 0, it gets the relevant reward when it reaches level 2. The reward is positive for moving left and negative for right. The test is interesting because it means rewards overlap; the reward received immediately after executing an action is actually due to the previous action.

9.1.3.1 Experimental Protocol

We applied IState-GPOMDP, with lines 7 and 8 of Algorithm 2 replaced by Equation (9.1), to all tests. The gradient estimation time used for each test is shown in Table 9.1.

The bias optimal filter for Test I, where $p = 0$ in Figure 9.2, has a finite response with impulses corresponding to the three possible reward delays $\tau = 2$, 5 and 8. This

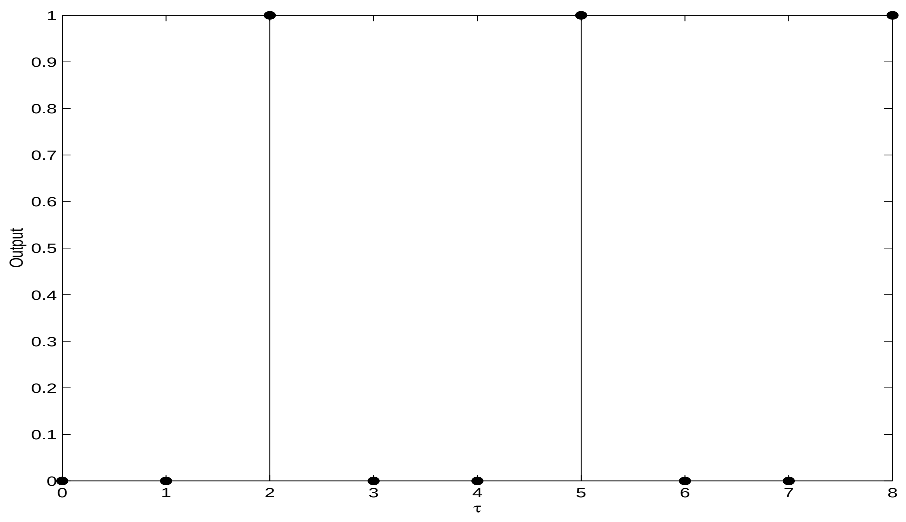


Figure 9.4: The optimal FIR filter for Test I, with parameters $a = [1]$, $b = [0, 0, 1, 0, 0, 1, 0, 0, 1]$.

filter is shown in Figure 9.4.

We applied two filters to Test II. The first is an FIR filter that assumes that rewards must be received between 2–12 steps after the relevant action. It makes no other assumption so all impulses between $\tau = 2$ and $\tau = 12$ have equal value, defining a rectangular filter shown in Figure 9.5. A good filter for Test II, where $p = 0.5$, should have an infinite response since rewards can be received an arbitrarily long time into the future. We tested an IIR filter with impulses at the same places as Figure 9.4, but the impulses were allowed to decay exponentially by setting the a_1 weight to -0.75. This filter is shown in Figure 9.6. It might be suspected that we should decay the impulses by a factor of p, however we found empirically that this produced a bias and variance worse than the FIR filter. Intuition, and our early experience, indicates that it is important to over-estimate credit assignment if bias needs to be minimised.

The optimal filter for both Tests III and IV is simply $a = [1]$ and $b = [0, 1]$, a single impulse at $\tau = 1$. This takes account of the fact that all rewards are delayed by 1 step.

Because our test POMPDs are small, we can compute true gradient by evaluating Equation (4.6) directly. The true gradient was compared to 50 IState-GPOMDP-IIR gradient estimates for each Test and filter. We also compare the estimates produced by IState-GPOMDP with exponential discounting of $\beta = 0.9$ and $\beta = 0.99$.

9.1.3.2 Results

The bias and variance of the estimates are shown in Table 9.1. For Test I, $\beta = 0.9$ produced a gradient pointing in the wrong direction; $\beta = 0.99$ is in the correct direction

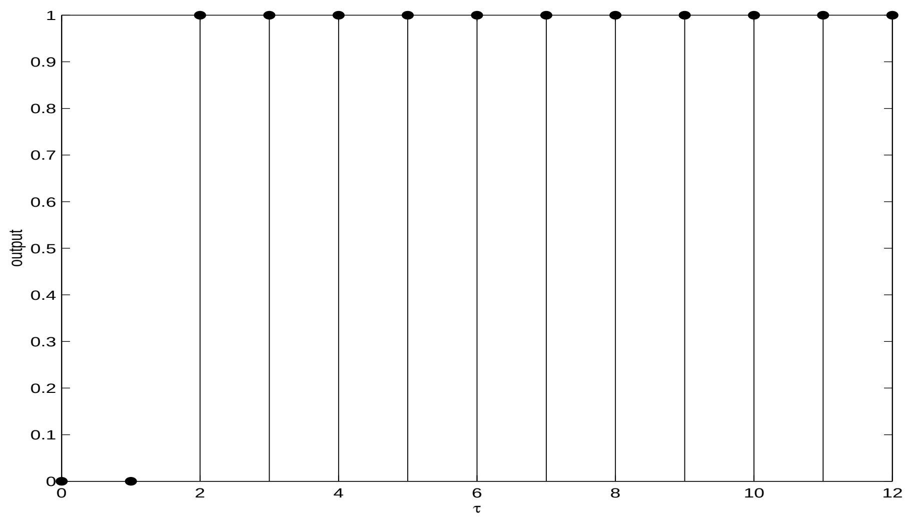


Figure 9.5: A good FIR filter for Test II, with parameters $a = [1]$, $b = [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$.

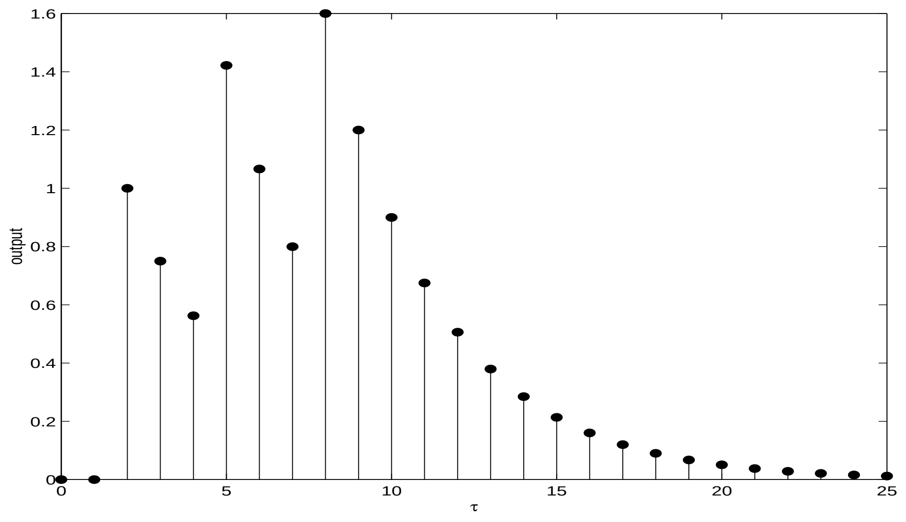


Figure 9.6: A good IIR filter for Test II, with parameters $a = [1, -0.75]$, $b = [0, 0, 1, 0, 0, 1, 0, 0, 1]$.

Table 9.1: Results of eligibility trace filtering experiments over 50 runs. There is a large drop in bias and variance for most FIR and IIR runs.

T	Test I 10^6		Test II 10^6		Test III 1000		Test IV 400	
Trace type	*Bias*	*Var.*	*Bias*	*Var.*	*Bias*	*Var.*	*Bias*	*Var.*
$\beta = 0.9$	176°	12.3	176°	18.4	0.610°	0.560	1.11°	111
$\beta = 0.99$	14.7°	2090	14.7°	2140	1.15°	2.88	2.36°	655
FIR	0.107°	7.72	13.9°	10.71	0.0450°	0.278	0.394°	16.7
IIR			4.35°	59.5				

but the high variance meant a total of around 1,000,000 estimation steps were required to achieve convergence to the correct policy. The simple FIR filter required only around 10,000 estimation steps.

In Test II, the FIR filter only marginally improved the bias, however the variance was greatly reduced. The IIR filter improved the bias further because it does not arbitrarily cut off credit after 12 steps, but introducing an infinite response increased the variance. This demonstrates that the bias/variance tradeoff in the choice of discount factor is still evident when designing arbitrary filters. Theorem 8 tells us that *one* unbiased filter for any POMDP is an infinite step response. A large class of POMDPs have unbiased filters that are not infinite step responses. For example, the POMDP of Test I and any POMDP that visits a recurrent state after at most T steps. Trivially, a zero-variance filter for all POMDPs has a 0 impulse response. Thus, the design of optimal filters requires fixing the maximum tolerable bias, or variance, before hand. Alternatively, we could design suitable metrics that trade off the bias and variance in a sensible way.

Tests III and IV also show an order of magnitude improvement in bias and variance.

We emphasise that this form of trace filtering can be applied to any RL algorithm that uses an eligibility trace, such as $\mathsf{TD}(\lambda)$ or $\mathsf{SARSA}(\lambda)$. A drawback of arbitrary filters is that the time taken to update the eligibility trace grows linearly with the number of filter-coefficients. Better results may be achieved with more steps of a simple high-variance filter than less steps of a complex low-variance filter. The speech recognition experiments of Section 10.3.2 applies eligibility trace filters to a real-world problem.

9.2 GPOMDP-SARSA Hybrids

Another variance reduction scheme we investigated is using value-function methods to learn the policy parameters θ while using IState-GPOMDP to learn the FSC parameters ϕ. Thus, we attempt to balance the convergence to a local maximum guarantees of policy-gradient methods with the lower variance of value methods (see Section 2.5.8).

We achieve variance reduction in two ways: (1) there are fewer gradients to be estimated because we only estimate the ϕ parameter gradients; (2) empirical evidence suggests that value-methods, if they converge at all, often converge more quickly than policy-gradient methods. This means less time will be wasted exploring irrelevant regions of the state space, giving the gradient estimator a boost from the improved sample relevance.

Our choice of SARSA(λ) for the value-method is based on empirical results that suggest that SARSA can often cope well with partial observability [Loch and Singh, 1998]. The algorithm details and some favourable preliminary results can be found in Appendix C.1. We will not discuss this approach further in the main thesis.

9.3 Importance Sampling for IState-GPOMDP

For some domains agents can be easily designed that perform much better than a purely random agent. For example, it is reasonably easy to code a controller for a robot that tells it that running into an obstacle is bad. A hand coded controller that does just this, combined with Q-learning, is discussed by Mitchell and Thrun [1996]. The hand designed agent is referred to as the *teacher*. The goal is to learn a better agent than the teacher, but in much less time than starting with a random agent. One way to do this is to initialise the learning agent with the teacher's policy, but this may put the agent in a local maximum. Also, it may not be easy to initialise an agent such as a multi-layer neural network with the correct policy. Importance sampling (IS), introduced in Section 2.7.1.1, provides a statistically well motivated mechanism for using a teacher to guide the learning agent.

IS has been used in conjunction with Monte-Carlo techniques as a method of reducing variance [Glynn, 1996]. Section 2.7.1.1 describes how IS weights the importance of each sample of an unknown distribution by the probability mass associated with the known sampling distribution. In the policy-gradient arena, hard-wired controllers have been used to implement the teacher policy $\tilde{\mu}(u|h, y)$ to help Williams' REINFORCE to estimate gradients with a minimum number of world interactions [Meuleau et al., 2000]. The teacher $\tilde{\mu}(u|h, y)$ serves as the known distribution which is used to generate actions, and the policy $\mu(u|\theta, h, y)$ is considered the unknown distribution. IS can also be used to implement *off-policy* methods (see Section 2.4) because IS lets us adjust the gradient contribution of each step $\mu(u|\theta, h, y)$ to account for the fact that a policy other than $\mu(u|\theta, h, y)$ is generating the actions [Peshkin and Shelton, 2002].

IS is a candidate method for reducing the variance of most of the algorithms developed in this thesis. The rest of this section shows that the implementation of IS is non-trivial for our algorithms, explaining why we have not implemented it for any of our experiments.

Consider adding IS to memory-less IState-GPOMDP. Assume we can construct the teacher's policy $\tilde{\mu}(u|y)$ before training begins. Then the following theorem holds concerning the gradient estimate with respect to the policy parameters θ.

Theorem 9. *Let $\tilde{\mu}(u|y)$ be the probability of the teacher selecting action $u \in \mathcal{U}$ after observing $y \in \mathcal{Y}$. Let $\tilde{\pi}$ be the unique stationary distribution of the underlying MDP under the teacher. The distribution $\tilde{\pi}$ may or may not depend on θ, but the stationary distribution under the agent, π, always depends on θ. Under the usual assumptions for IState-GPOMDP, and the assumption that $\frac{|\nabla\mu(u|\theta,y)|}{\tilde{\mu}(u|y)}$ and $\frac{\pi(i|\theta)}{\tilde{\pi}(i)}$ exist and are bounded $\forall\theta, y, u, i$, we have with probability 1*

$$\pi'(\nabla P)J_\beta = \lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\frac{\nabla\mu(u_t|\theta,y_t)}{\tilde{\mu}(u_t|y_t)}\frac{\pi(i_t|\theta)}{\tilde{\pi}(i_t)}\sum_{s=t+1}^{T}\beta^{s-t-1}r_s. \tag{9.2}$$

This equation, derived in Appendix A.5, cannot be implemented directly due to the dependence on the unknown ratio $\pi(i|\theta)/\tilde{\pi}(i)$. Although $\tilde{\pi}$ is not shown to be a function of θ, it may depend indirectly on the parameters. For example, we can make $\tilde{\mu}(u|\theta, y)$ a mix of a teacher policy followed with probability ϵ, and $\mu(u|\theta, h, y)$ followed with probability $1 - \epsilon$.

The analysis above applies equally to IState-GPOMDP and Exp-GPOMDP. The problem arises due to the infinite-horizon setting. However, the theorem does not completely preclude the possibility of using IS with IState-GPOMDP. Further work will investigate eliminating the unknown ratio through Monte-Carlo methods, similar to the way the stationary distribution in $\pi'\nabla PJ_\beta$ is eliminated while deriving the IState-GPOMDP algorithm (as shown in Appendix A.2.2).

9.4 Fixed Random Number Generators

Section 2.7.1.3 introduced the idea of using fixed random number generators during simulation to reduce the variance of Monte-Carlo value estimates. The trajectory of a POMDP simulated on a computer is a deterministic function of the current policy and the computer's random number generator. Using a fixed sequence of random numbers guarantees the same policy will always generate the same estimate of η, even when the world-state transitions $q(j|i,u)$, and observations $\nu(y|i)$, are not deterministic. Suppose we have two policies, μ_a and μ_b, with long-term average rewards η_a and η_b. We estimate both average rewards with the same random number sequence. If $\eta_a > \eta_b$ the difference *must* be due to the difference between policies μ_a and μ_b. The difference cannot be due to policy μ_a experiencing a "lucky" trajectory through the POMDP.

9.4.1 Application to Policy-Gradient Methods

Comparing policy values using m sample trajectories, generated with m sequences of random numbers, is the basis of the PEGASUS algorithm [Ng and Jordan, 2000]. For infinite-horizon scenarios we use one long sequence of random numbers instead of m short sequences. Applying this trick to policy-gradient algorithms ensures that differences in the gradient estimates are due to the differences between policies. This is helpful when comparing gradient estimates during the line search phase of gradient ascent. The line search detailed in Appendix B.1 computes the sign of the dot product between the search direction and a local gradient estimate at a point along the search direction. If the sign is negative it means the local gradient and the search direction disagree on which direction is uphill by more than 90°. By fixing the random number generator we ensure that the sign of the dot product changes because of changes to the policy, and not because of the particular trajectory that was followed.

The practical implication is a reduction in the number of simulation steps needed to achieve consistent gradient estimates. Without a fixed random number generator we must keep simulating long enough to wash out the effect of following a particular trajectory. Unfortunately, consistent gradient estimates do not imply correct gradient estimates. Gradient estimates based on the same random sequence, but too short an estimation time, will be mutually consistent but may be more than 90° from the true gradient.[2]

9.4.2 Caveat: Over-Fitting

Although using fixed random number sequences is an effective method for reducing the variance of simulated Monte-Carlo methods, we discovered that this method can introduce spurious local maxima for both policy-gradient and value-function methods. The effect is a form of over-fitting. Because the agent is trained on a finite sequence

[2]A trick for determining appropriate estimation times is to deliberately use different random number sequences to generate gradient estimates under the same policy. If the sign of the dot product between these estimates is negative, then the estimates differ by more than 90° and the estimation time must be increased. Estimation times were chosen this way for our experiments when we describe the chosen T as the smallest consistent gradient estimate time.

of random numbers, it can learn to take advantage of the *particular* random sequence being used, rather than the true distribution of POMDP state trajectories.

When the true gradient is weaker than the effects of the fixed random sequences the estimated gradient can lead the agent into one of these spurious maxima. Consider the Heaven/Hell scenario of Section 8.2. If we use a fixed random sequence that results in the agent being placed in heaven-left world 51 times, and the heaven-right world 49 times, then the agent can learn to achieve a small but consistently positive reward by always moving left at the intersection. This effect was observed in practice and the result can be seen in Table 8.2. Even the dense FSC results for IState-GPOMDP are consistently positive (though very small). The agents learnt a policy that always moves left or right depending on the initial seeding of the random number generator.

Appendix C.2 discusses this phenomenon in greater detail, providing a simple example with experimental results. To avoid the problem, both in value-based approaches and policy-gradient approaches, we can increase the number of random numbers used or periodically re-generate random sequences. The latter option is equivalent to periodically changing the seed of the random number generator.

9.5 Summary

Key Points

⊶ I If we have prior knowledge about temporal credit assignment we can encode this in an IIR filter to reduce the bias and variance of gradient estimates.

⊶ II Low-variance value based methods such as SARSA can be used to learn θ, while policy-gradient methods like IState-GPOMDP learn ϕ. The use of SARSA to learn θ has a variance and bias reducing effect on IState-GPOMDP.

⊶ III The PEGASUS trick of using fixed random sequences is useful for policy-gradient methods but can cause over-fitting, resulting in sub-optimal agents.

Future Work

It would be useful to design optimal IIR filters given a description of the delay between actions and rewards. To do this we need metrics that trade off bias and variance, or we can minimise the variance subject to the constraint that the bias must be below some threshold.

While the experiments for GPOMDP-SARSA in Appendix C.1 are promising, we have not yet demonstrated that the algorithm can lead to reduction in the number of world-interactions needed. Because GPOMDP and SARSA use independent Monte-Carlo samples it may require more total samples than plain IState-GPOMDP, which uses the same sample to learn ϕ and θ simultaneously.

Applying importance sampling to infinite-horizon POMDPs is also a fascinating avenue for future research.

Chapter 10

Policy-Gradient Methods for Speech Processing

For every complex problem there is a solution which is simple, neat and wrong.
—Henry L. Mencken

In this chapter we apply some of the algorithms and methods from previous chapters, gradually working up to a difficult real world problem: large vocabulary continuous speech recognition (LVCSR). We start by demonstrating that signal classification can be cast as a POMDP and that policy-gradient trained agents have classification accuracy comparable to those of an hidden Markov model (HMM). We then show that policy-gradient methods can be applied to real speech problems, solving a simple spoken digit classification and segmentation task. We end by training a phoneme level LVCSR system, showing that POMDP agent training can achieve better results than training the same agent to maximise likelihoods.

Although treating signal classification in the POMDP framework will be shown to have some benefits, we do not claim that the POMDP model will outperform the current state-of-the-art in speech recognition. Instead, we wish apply POMDPs to a real world problem, in the process investigating a novel and interesting approach to speech processing.

Throughout this chapter a basic understanding of speech technology is helpful but a full review is out of the scope of the thesis body. An introduction to HMMs, the Baum-Welch training procedure, and Viterbi decoding is contained in Appendix D. A literature survey of advanced connectionist and discriminative speech processing methods, relevant to this Chapter, is contained in Appendix E.

10.1 Classifying Non-Stationary Signals with POMDPs

The actions chosen by the agent need not affect the environment directly. Alternatively, we can view the action chosen at each time step as being a label, or classification, of the observations being produced by the environment. For example, the signal may come from a radio telescope. At each time step the observation is a representation of the signal at that instant. The action is a decision about whether an alien intelligence sent the signal. Internal state is required to remember enough about the signal history to make a good decision.

A key advantage of reinforcement learning compared to other sequential data algorithms, such as HMMs or recurrent neural networks (RNNs), is that we can define an arbitrary reward signal to be maximised. This allows us to maximise the most relevant reward function, instead of being limited to maximising likelihoods or minimising the mean square error. For our example above, we could train the system with a mix of signals from the Earth and empty space, rewarding the controller when it makes the right classification. This goal function is intuitively correct since it aims to maximise our definition of performance directly. Alternatively, if we used maximum likelihood training of HMMs, one problem that could occur is that a single spurious low probability signal could introduce errors for a long period of time, sometimes described as a "weak link" in the Markov chain. Directly maximising the quantity we care about may avoid such problems by generalising better to low probability events, or ignoring events that are irrelevant to overall performance.

By treating signal processing as a POMDP we enter a planning domain rather than a static pattern recognition domain. The planning domain allows the agent to make decisions about what information to remember, and what information to ignore; possibly asking for more information if necessary. This ability is useful in the speech domain where there is a wealth of features. Some features, such as visual cues, are asynchronous and spread across different time scales.

Before focusing on comparing HMM and POMDP signal classification, we discuss another method which can discriminate non-stationary, variable length signals. Jaakkola and Haussler [1998] use a kernel function derived from generative methods to perform classification. Generative models such as logistic regression and HMMs can be used to construct the kernel. The kernel is then used to build a classifier using support vector machines or other kernel-based methods. The kernel function computes the inner product of the gradients of probabilities from the generative model. For example, in the first step a signal is processed by an HMM and the gradient of the signal probability given the HMM parameters is computed. The kernel function then generates a distance between two signals by computing the inner product of the gradients invoked by the different signals. The method is similar to IState-GPOMDP only in that both methods seek a *discriminative* method for signal classification. However, Jaakkola and Hausler's method relies on estimating the likelihood of a signal using previously estimated generative models. Our approach attempts to bypass the estimation of probabilities and does not generate distances between one signal and another.

Since HMMs are the popular choice for speech processing, the rest of this section compares IState-GPOMDP to HMM training. To begin with, HMM methods train individual HMMs only on the subset of the data that they model. On the other hand, IState-GPOMDP is inherently discriminatory: the agent is forced to choose the best classification based on the differences between the signal types rather than its ability to model each signal type independently.[1] For example, a simple speech recognition system might train an HMM with 3 to 5 states for each phoneme in the language. This implies, at least for initial training, that the training data is labelled and split into a collection of instances of each phoneme. Each HMM is trained only on the collection of instances for the phoneme it models. During classification the HMM that has the highest likelihood is used to identify the best phoneme match. Empirically, this has been observed to lead to high misclassification rates between similar phonemes since the classifier does not take into account the relative differences between phonemes [Lee et al., 1995]. IState-GPOMDP trains a single agent on all the data, so classification can be performed by concentrating on the distinguishing features of the phonemes.

Also unlike HMM training, IState-GPOMDP does not immediately assume that local sections of speech (corresponding to a single HMM state) are stationary, so we could automatically learn to cope with effects such as the co-articulation of phonemes in speech [Lee, 1989]. HMMs also assume that the amount of time spent in each state follows an exponentially decaying distribution [Rabiner, 1989]; IState-GPOMDP trained models can potentially model different speech durations by altering the I-state transition distributions according to the current observation. Finally, IState-GPOMDP is used to train a single agent using the entire data set, reducing the problems encountered when training models with only the small volume of data relevant to each model.

Our initial experiments use simple HMMs to generate signals, which we then attempt to classify using an IState-GPOMDP trained agent. Subsequent experiments extend this to a simple binary speech classification problem followed by the LVCSR experiments.

10.2 Classifying HMM Generated Signals

Since HMMs can model non-stationary signals we use them to generate data with which to train and test FSC signal classification. The accuracy of HMM methods for classifying the same test data gives us a baseline with which to compare FSC agents to HMM classification.

[1]Discriminative training is sometimes used for state-of-the-art HMMs, described in Appendix E. More generally, a large body of literature is devoted to modifications of HMMs which attempt to overcome the limitations described in this Chapter. For example, the problems HMMs have modelling fine-scale co-articulation of phonemes has been investigated by Deng [1998].

Assume an environment that implements HMMs $\mathcal{M} = \{1, \ldots, |\mathcal{M}|\}$. Each HMM $m \in \mathcal{M}$ is defined by a state transition matrix and discrete emission distribution matrix, though we do not refer to these explicitly in this section. We generate a sequence of symbols y_t from model m and after a random period switch to another model m'. We want the agent to emit actions $u_t = m$ for observations y_t from m, and then switch to $u_t = m'$ for observations y_t from m'.

10.2.1 Agent Parameterisation

Our classification agents are based on the same kind of FSCs and policies trained for the experiments of Chapter 8. We now describe how $\omega(h|\phi, g, y)$ and $\mu(u|\theta, h, y)$ are parameterised for the remainder of the experiments in this section. We tried two parameterisations of $\omega(h|\phi, g, y)$, and one for $\mu(u|\theta, h, y)$.

10.2.1.1 Policy Parameterisation

We make $\mu(u|\theta, h, y)$ a deterministic function of the current I-state so that the I-states are evenly split into $|\mathcal{M}|$ groups, with each group assigned to always emit labels for one of the possible models

$$\begin{aligned} u^*(h, |\mathcal{M}|) &= h \bmod |\mathcal{M}|, \\ \mu(u|\theta, h, y) &= \chi_u(u^*). \end{aligned} \tag{10.1}$$

The main reason for this choice is that we had not properly analysed the zero-gradient regions discussed in Chapter 7 prior to running these experiments, so performance was poor when we also tried to learn a parameterised $\mu(u|\theta, h, y)$. As illustrated by Figure 7.2, stronger gradients are generated by starting with a fully connected $\omega(h|\phi, g, y)$ and deterministic $\mu(u|\theta, h, y)$, than starting with a sparse FSC and stochastic $\mu(u|\theta, h, y)$. For this reason we expect faster convergence using Equation (10.1) than using the sparse FSCs applied in the experiments of Chapter 8. The drawback is that the final performance of the classification agent may be poorer than could be achieved by learning $\mu(u|\theta, h, y)$. One reason for this is that we have assumed that the model labels should be uniformly associated with I-states. If it turns out that one particular model is harder to estimate than others, then perhaps more of the I-states should be associated with that model.

10.2.1.2 FSC Parameterisation

The lookup table parameterisation of Section 3.4.1 requires $|\mathcal{G}|^2|\mathcal{Y}|$ parameters to represent $\omega(h|\phi, g, y)$. If we wish to use many I-states or n-dimension real inputs $y_t \in \mathbb{R}^n$, the lookup table falls prey to the curse of dimensionality. For real-world problems we need to use function approximators such as ANNs. We experiment with both lookup tables and ANNs in the experiments of this section. Following this section we use ANNs exclusively because we will be working with real-valued, multi-dimensional acoustic feature vectors.

For the ANN experiments we use the neural network described in Section 3.4.2 and shown in Figure 3.1. The I-state g_t becomes an input feature by augmenting the network inputs used to encode the observation with $|\mathcal{G}|$ extra input nodes using 1-in-n encoding. At each step, input g_t is 1 and the remainder are 0. Assuming n_h hidden units, this model requires $n_h(|\mathcal{Y}| + 2|\mathcal{G}|)$ parameters and needs no modification to accept real–valued observations. Section 3.4.2 describes how gradients of $\omega(h|\phi, g, y)$ are computed.

All the ANN experiments reported in this chapter use a fast matrix-matrix multiply, developed for this task, which uses the 4-way parallelism of the Intel Pentium III and more recent CPUs. Fast ANN training is discussed in Chapter 11.

10.2.2 Classifying Pre-Segmented Signals

Four simple tests were hand-crafted to investigate FSC agent classification of signals produced by $|\mathcal{M}|$ different HMMs. The aim was to determine if FSC classification can be competitive with HMM based methods.

For these tests we assume that segments of observations y_t, emitted by the source HMMs, are of known length and generated by a single model. The agent receives a reward of 1 for issuing the correct label and -1 for an incorrect label. Thus, the long-term average reward η is a measure of how many labels are correct on average.

The FSC agents classify *segments* (as opposed to observations) by choosing the model corresponding to the most often emitted label u_t, as chosen using Equation (10.1), over the duration of the segment. If the correct action is chosen on average, the decision will be correct. The FSC classification agent is depicted in Figure 10.1. We apply HMM training methods for comparison. They perform classification by choosing the model with the highest likelihood of matching the segment of observations y_t.

Each test discriminates between 2 or 3 models, each with 2 symbols and 2 states. This may seem easy, however, Tests I–III are all sets of source HMMs with exactly the same stationary distribution of symbols. Only by learning the *temporal* or *non-stationary* characteristics can a classifier discriminate between signals produced by these HMMs. Figure 10.2 shows the HMMs used for generating sequences for Test I. They are identical except one model prefers to stay in the same state, and the other prefers to alternate states. The difficulty of the task is illustrated by Figure 10.3, a

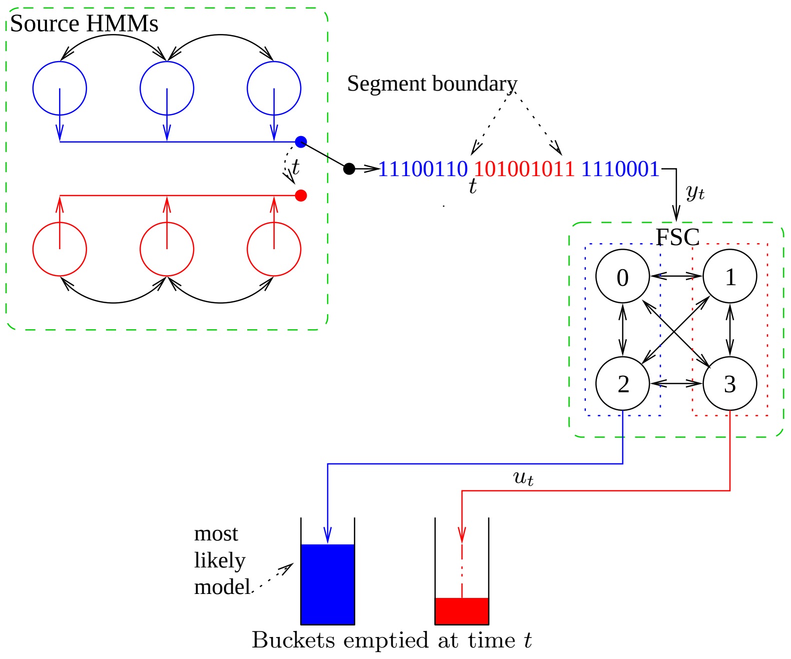


Figure 10.1: An illustration of the FSC agent that discriminates between sequences generated by the blue and red HMMs. If the FSC is in state 0 or 2, the emitted action indicates the blue HMM generated the signal, otherwise it indicates the red HMM generated the signal. The blue and red buckets count the number of actions labelling each source. At time t, the end of the segment, the fullest bucket indicates the final classification of the source HMM. Then the source HMM is re-chosen and the buckets emptied. If the segment length is not known the bucket mechanism cannot be used.

graphical version of the 2 signals generated by the models of Test II. The details of the HMMs used in each case can be found in Appendix F.1.

The test cases respectively represent:

I. the ability to discriminate between two simple signals with the same stationary distribution;

II. a harder version of test I;

III. the ability to discriminate between 3 models (see Appendix F.1 for the models);

IV. the ability to discriminate between signals with different stationary distributions. Such signals can be classified by counting the occurrence of each symbol.

10.2.2.1 Experimental Protocol

We trained 4 classifiers on each test case and compared them to the classification performance of the source HMMs. The 4 classifiers are:

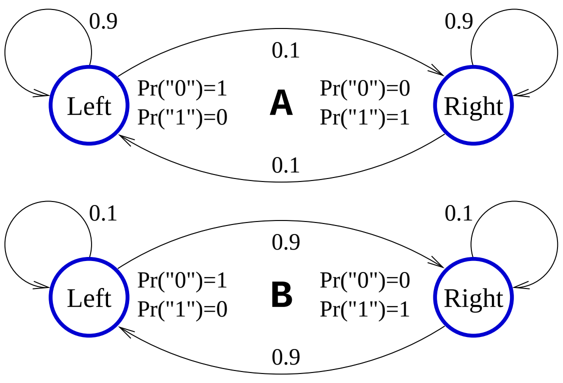


Figure 10.2: HMMs used to generate sequences for Test I and to generate baseline classification results. Test II is the same except that emission probabilities become 0.9 and 0.1 instead of 1.0 and 0.0 respectively.

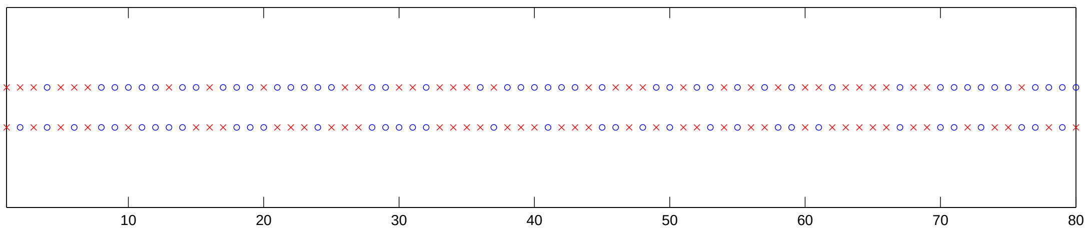


Figure 10.3: Sequences generated by the models for Test II. Which is which? The upper line is model `A` and the lower is `B`

- An IState-GPOMDP trained agent using a lookup table for $\omega(h|\phi, g, y)$, and the deterministic policy $\mu(u|h)$ described in Section 10.2.1.1. Classification is performed by feeding in a segment of known length and counting the number of instances of each label emitted as an action. The most frequent label identifies the source HMM.

- An IState-GPOMDP trained agent using a single layer (linear) ANN for $\omega(h|\phi, g, y)$ as described in Section 10.2.1.2, and deterministic $\mu(u|h)$. Classification is again performed by counting the labels output over the duration of a segment.

- A discrete IOHMM trained using Algorithm 9 (Appendix D.2). In this case the observations y_t drive the transitions, and the IOHMM emissions are model labels. Classification is performed by computing the forward probability (D.9) of the IOHMM model when the IOHMM emission sequence is a continuous stream of the hypothesised label. For example, suppose the input sequence was $\{y_t\} = \{0, 0, 1, 1\}$, and we must classify it as being generated by model `A` or `B` from Figure 10.2. We compute the forward probability of the IOHMM with the hypothesised emission sequence {`A, A, A, A`} and {`B, B, B, B`}, picking the result with the highest likelihood as the best classification. We would expect that the example sequence was probably generated by model `A`.

- Classical HMMs trained by the Baum-Welch procedure (see Appendix D.1.1). An HMM is trained for each source model. The HMMs are trained so that they emit the observation sequence $\{y_0, \dots, y_t\}$ generated by the associated source model. There is no driving sequence in this case. Classification is performed by computing the forward probability (D.1) and the Viterbi probability (D.8) for each model and picking the model with the highest likelihood. The forward probability gives the true likelihood of the model given the segment. Since the Viterbi probability is an approximation to the forward probability, we expect the Viterbi results to be slightly worse than the forward probability results. We would have to use the Viterbi probability for cases in which one long signal from multiple models needs to be split into segments that belong to each model.

The IState-GPOMDP trained classifiers were trained with $|\mathcal{G}| = 2, 4 \dots, 24$ I-states. Table 10.1 shows how many I-states gave the best performance for each test. In the cases of the lookup table and the ANN, the parameters ϕ were initialised randomly in the range $[-0.5, 0.5]$. The IOHMM transition matrices were initialised to uniform distributions. The emission distribution matrix elements were initialised randomly between $[-0.1, 0.1]$, then 0.5 was added to all elements before each row was normalised. This has the effect of producing distributions that are slightly perturbed from uniform, as suggested by Rabiner [1989]. The HMM experiments were performed with `UMDHMM` v1.02 [Lee, 2000]. It is not clear how `UMDHMM` initialises the HMM parameters.

The discount factor was set to $\beta = 0.9$, reflecting the fact that the reward should only be influenced by relatively recent I-state transitions, and the quadratic penalty was set to $\wp = 0.001$. Recall from Section 4.5.1 that the quadratic penalty for all experiments was set so that the maximum parameter weight does not surpass 0.5 until the penalty is reduced for the first time. Trial and error led us to select a gradient estimation time of $T = 5 \times 10^6$ steps for the IState-GPOMDP trained agents, and a line search gradient estimation time of 1.25×10^6 steps.

Training data was generated by running each model in $\mathcal{M}$ for 100,000 steps, using `UMDHMM`, and recording the observations $\{y_0, \dots, y_{99,999}\}$. For IState-GPOMDP, segments of random length, between 20–200 observations, were sampled from the data for a randomly selected model. These segment length bounds were selected to reflect the typical number of acoustic vectors per phoneme that are available for speech applications. Once IState-GPOMDP consumes all the observations in a segment a new segment is sampled, providing an infinite stream of segments, fitting into the infinite-horizon POMDP setting.

Training data for estimating new HMMs was the raw 100,000 step observation sequences. HMM models were trained only on the data generated for that model, that is, training was not discriminative.

IState-GPOMDP training completes in about 1 hour using a single 550 MHz Pentium III CPU. HMM training took approximately 3 minutes per model.

Test data was produced by generating 1000 sequences of length 20–200 for each model. Thus, for Tests I, II, and IV, there were 2000 segments to be classified, and Test III had 3000 segments. In the POMDP agent case, segments were presented one at a time, resetting the I-state to a random value before each classification run.

Classification using the *source* HMM models and the forward probability produces the optimum maximum likelihood (ML) classifier because the true models are being used to evaluate the probability of generating the segment. Since we use a uniform prior on the models, the likelihoods give us a scaled version of the maximum posterior probability that the model fits the data. The *trained* HMMs indicate the level of performance we should expect without knowledge of the source HMMs. The lower accuracy of the trained HMMs compared to the source HMMs illustrates the effect of training with limited data.

10.2.2.2 Results

Table 10.1 shows the percentage of misclassified segments for each test and classifier architecture. The "$|\mathcal{G}|$" column shows how many I-states were used to achieve the result in the "Error" column. The "For" columns are the results generated using the forward probability calculation and the "Vit" columns are the results generated using the Viterbi approximation to the forward probability.

As expected, the source "ML HMM"s gave the best results in all tests. In general though, we see that IState-GPOMDP was competitive with results generated using trained HMMs and Viterbi decoding, which is a fair comparison since the agents emit a string of signal classifications just as the Viterbi procedure does. Also as expected, the lookup table parameterisation of $\omega(h|\phi, g, y)$ produced better results than the approximate ANN parameterisation.

Analysis of the results for Tests I & II show that IState-GPOMDP learnt the $|\mathcal{G}| = 4$ I-state policy graph illustrated in Figure 10.4. If the I-state is 1 or 3 at time t, then a label corresponding to model `A` is emitted, otherwise the label for model `B` is emitted. The graph represents the policy: *"if we see the same observation twice in a row, emit `A`, otherwise emit `B`."*

Table 10.1: Percentage of HMM segments misclassified for each test. "Lookup table" and "ANN" are the agents trained with IState-GPOMDP. The "ML HMM" column gives the results of using the *source* HMMs to classify segments. The "HMM train" column gives the results of training $|\mathcal{M}|$ HMMs on limited data.

Test	*lookup table*		*ANN*		*IOHMM*		*ML HMM*		*HMM train*	
	Error	$\|\mathcal{G}\|$	Error	$\|\mathcal{G}\|$	Error	$\|\mathcal{G}\|$	For.	Vit.	For.	Vit.
I	0.0	4	0.0	4	0.0	4	0.0	0.0	0.0	0.0
II	5.5	4	8.9	4	4.3	4	0.9	2.8	4.6	4.8
III	0.1	12	1.5	10	5.1	12	0.0	0.0	0.0	1.0
IV	0.4	12	3.0	12	10.6	12	0.1	0.1	0.3	0.4

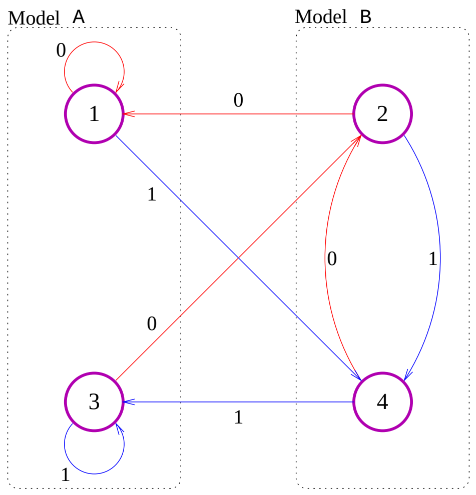


Figure 10.4: The policy graph for Tests I & II learnt using IState-GPOMDP. The left I-states emit label A and the right I-states emit label B. I-state transitions are labelled with the observations that cause those transitions.

This policy graph demonstrates that the agent has learnt a discriminative classifier that did not maximise likelihoods as an HMM would. No matter what the previous statistical evidence is, the last two observations dictate which label will be emitted, which maximises the correctness of the average label decision. The fact that the agent learnt a deterministic FSC tells us that it generated a different function than an HMM would. An HMM with deterministic transitions and emissions cannot generate stochastic signals such as our source signals. As soon as a deterministic HMM encounters an unexpected symbol, the model probability drops to 0.

10.2.3 Classifying Unsegmented Signals

In this case segments of an unknown number of observations y_t are generated by different models. The task is to detect the boundary at which the observation source switches from one model to another. When this occurs the average action should switch to the correct label. A local average of the labels can be computed by filtering the label sequence with a rectangular window. The rectangular filter decision function is

$$m_t = \arg\max_u \sum_{s=t-\frac{w}{2}}^{t+\frac{w}{2}} \mu(u|h_s), \tag{10.2}$$

where w is the window length and tunes the sensitivity of the classification to changes in model. The more frequently the model changes, the lower w should be. Because the filter looks $w/2$ steps into the future, the final decision for each step is delayed by $w/2$ steps. Triangular and other windows that weight the current label more heavily may seem more desirable, however, since our goal function is to maximise the average correctness of the label, and not just the current label, a window that weights all local decisions equally best fits our goal function.

Alternatively, we could *learn* $\mu(u|\theta, h, y)$ and use it as the likelihood of a label given the current I-state. The Viterbi algorithm, described in Appendix D.1.2, can then be applied to choose the most likely sequence of actions (hence labels). Applied in this way, the classification procedure closely resembles ANN/HMM hybrids such as those described by Bourlard and Morgan [1998] and Bengio et al. [1992], however the training method is very different. We will explore this idea further for the LVCSR experiments in Section 10.3.

Our first experiment demonstrating the use of a rectangular window to compute segment boundaries was in the same style as the previous set of experiments, using hand-crafted HMMs to generate signals to be identified. This experiment and its discussion are deferred to Appendix F.2.

To demonstrate that the system works on real speech signals, the next experiment trained an agent to discriminate between the spoken digits "1" and "6." Segmentation of the speech signal was performed using the simple local averaging scheme described above.

10.2.3.1 Experimental Protocol

We used 394 training digits from 111 male speakers in the Texas Instruments Digits database [Leonard and Doddington, 1982]. Another 50 digits were reserved for testing. Exactly half the digits were "1" and half were "6." Silence was stripped from the start and end of each digit before concatenating them together into a continuous stream of digits in random order. The observations y_t were acoustic feature vectors calculated every 10 ms over a 25 ms Hamming window, providing a total of 40,212 training frames (402 seconds of speech). The extracted features are typical of those used for continuous density HMMs [Paul, 1990]: 12 mel-cepstral parameters with a pre-emphasis factor of 0.97, plus log energy. The cepstral features were shifted to have zero-mean. The delta[2] and double-delta values of these 13 features are computed and added to the acoustic vector for a total observation dimension of 39. Features were computed off-line using the `HTK` V3.0 toolkit [Woodland, 2001].

The inputs were normalised to between $[-1, 1]$ for presentation to the ANN. There were 16 I-states and 10 hidden nodes squashed with the tanh function. Using trial and error we set the estimation time to $T = 10^7$, the line search estimation time to 2.5×10^6, $\beta = 0.9$ and $\wp = 0.04$.

10.2.3.2 Results

Training took 1 hour on the "Bunyip" Beowulf cluster using 64 Pentium III 550 MHz CPUs. Figure 10.5 shows how the resulting agent classified and segmented a small portion of the test data. The per time step labels were correct 89% of the time. We smoothed the per frame classifications using the rectangular window described by Equation (10.2) with a window width of $w = 20$. This window length corresponds to 0.2 seconds of speech. The smoothed decision closely followed the correct classification. Declaring a change in digit when the smoothed decision crossed 0 on the digit axis correctly identified all the changes in the digit being spoken.

When the same digit was presented consecutively, thresholding the smoothed decision did not indicate the presence of multiple digits. The result was an inability to differentiate between the same digit being spoken once or multiple times. To rectify this, future work could train the agent to emit an additional `digit boundary` label.

Because we used a deterministic $\mu(u|h)$, we know that the agent learnt to classify signals by using the FSC. Figure 10.5 shows that very few 10 ms frames are misclassified, and most of the misclassified frames occur at the start of each new digit. From this it seems likely that the agent learnt that the next label is most likely to be the same as the last label, showing that it might have learnt to internally smooth its decision.

Instead of pursuing these experiments by implementing a full spoken digit recogniser, we jumped to the large vocabulary continuous speech recognition experiments that are the subject of the next section.

[2]Delta values take the difference between cepstral parameters over consecutive frames. Double-delta parameters take the difference of the differences over consecutive frames.

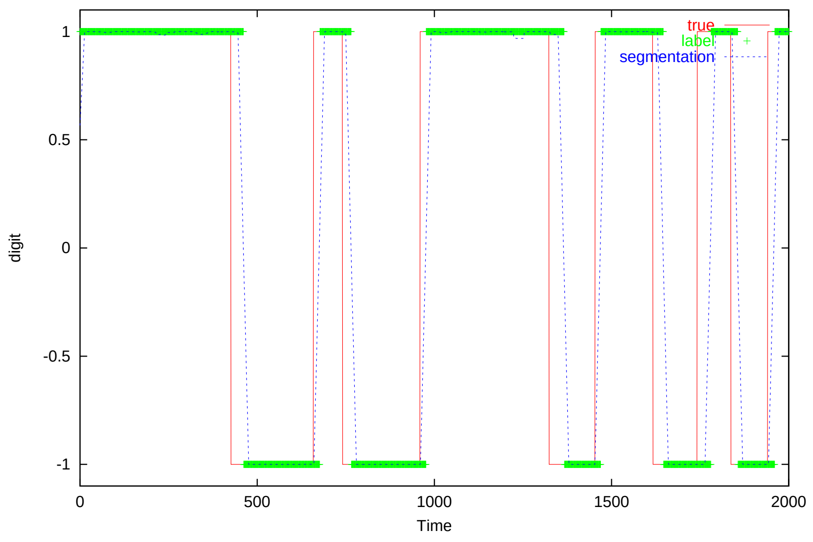


Figure 10.5: Classification of the digits “1” (1.0 on the plot) and “6” (-1.0). Although the smoothed decision closely follows the true classification, the system is not able to indicate when the same digit is spoken twice in a row, since adjacent digits do not force the classification to change.

10.3 Phoneme level LVCSR

In this ambitious experiment we attempt to train a low level phoneme classification system using policy-gradient methods. The heart of the system uses a specialised version of Exp-GPOMDP. Eligibility trace filtering is applied to reduce the variance of the gradient estimates. The resulting ANN/HMM hybrid is essentially the same as the architecture created by Bourlard et al. [1995], and expanded upon by Bourlard and Morgan [1998]. The relationship between this architecture and other connectionist speech technology is described in Appendix E.5.1. The novel aspect of our work is how we train the ANN/HMM hybrid.

Figure 10.6 describes the overall structure of an LVCSR speech recognition system. We construct all parts up to the output of a phoneme sequence, optimising the ANN and Viterbi components. We devote some time to describe the overall training procedure, along the way describing the operation of the ANN/HMM hybrid. Then in Section 10.3.2 we describe our experiments.

10.3.1 Training Procedure

The basic idea of the hybrid is to train the system in two phases: (1) a single large ANN is trained to estimate the emission probabilities of the HMM states; (2) these

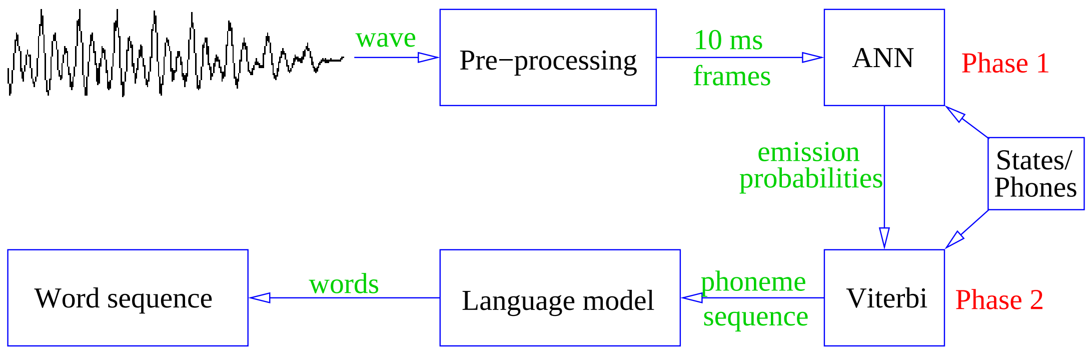


Figure 10.6: Block diagram of a complete speech recognition system. This section deals with the system up to the output of the phoneme sequence. The ANN and the Viterbi decoder are the subjects of Phase 1 and Phase 2 training respectively.

emission probabilities are used while estimating the HMM transition probabilities. Policy-gradient methods are used to perform both phases.

10.3.1.1 Learning Emission Likelihoods

Emission distributions measure the likelihood of acoustic vectors given that the HMM is in a particular state. Emission distribution estimation is normally done in parallel with training the HMM state transition probabilities, using an EM like procedure such as the Baum-Welch algorithm outlined in Section D.1.1. Modern systems model emission probabilities using a mixture of Gaussians for each HMM state [Paul, 1990].

Instead, we train an ANN to produce posterior estimates of HMM state probabilities, that is, $\Xi(m|y_t)$ where m is the acoustic unit and y_t is the acoustic observation vector from a frame of speech. We assume our acoustic units are phonemes. Thus, $\Xi(m|y_t)$ is the probability of phoneme m given the acoustic vector y_t. This probability does not take into account any previous acoustic vectors. It is easy to convert from a posteriori to a *scaled likelihood* by assuming $\Xi(y_t)$ is constant and using Bayes' rule

$$\xi(y_t|m) \propto \frac{\Xi(m|y_t)}{\Pr(m)}.$$

This simply amounts to dividing the network output probabilities by the phoneme priors $\Pr(m)$. The priors can be estimated from the training data. For the remainder of this chapter a phoneme m and an HMM state are considered synonymous because the HMM will consist of one phoneme per state. It is more common to model a phoneme using several HMM states, however our approach simplifies the task and follows the architecture constructed by Bourlard and Morgan [1994].

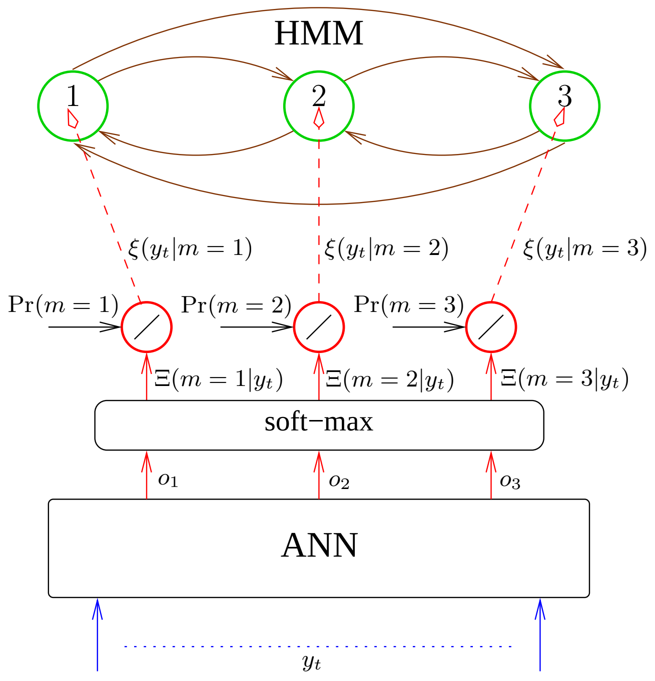


Figure 10.7: The ANN/HMM Hybrid model: using an ANN to generate HMM observation likelihoods.

The quantity $\xi(y_t|m)$ represents the emission likelihood of acoustic vector y_t assuming the current phoneme is m. Such likelihoods are used to model observation generation in HMM states. Figure 10.7 illustrates the hybrid process. The ANN provides the observation likelihoods that are needed to evaluate the likelihood of paths through the HMM states. We will briefly experiment with using multiple HMM states per phoneme, but we will assume that all HMM states for phoneme m have the same emission distributions, known as *tied distributions* [Lee, 1989].

The first stage of training uses memory-less IState-GPOMDP to train a large ANN to recognise phonemes based on frames of mel-cepstral acoustic vectors. The m'th network output filtered through the soft-max function estimates $\Xi(m|y_t)$, which is the posterior probability that the ANN input y_t is generated by phoneme m. In this case there is no internal state and the mel-cepstral features are treated as the POMDP observations y_t.

Let the correct label for frame t be denoted m_t^*. we now show that memory-less IState-GPOMDP with $\beta = 0$ and $r_{t+1} = \chi_{u_t}(m_t^*)$, estimates $\mu(u_t|\theta, y_t) = \Xi(m = u_t|\theta, y_t)$. This choice of reward function issues a reward of 1 when the action emits the correct phoneme label and 0 otherwise. From our proof in Appendix A.2.2 that IState-GPOMDP converges to an approximation of $\nabla\eta$ (see Equation (A.12)), or by unrolling the inner loop of Algorithm 2, we have with probability one as $T \to \infty$

$$\pi'(\nabla P)J_\beta = \frac{1}{T}\sum_{t=0}^{T-1} \frac{\nabla\mu(u_t|\theta, y_t)}{\mu(u_t|\theta, y_t)} \sum_{s=t+1}^{T} \beta^{s-t-1} r_s.$$

Defining $0^0 = 1$ and substituting for β and r_s obtains

$$\begin{aligned}\pi'(\nabla P)J_\beta &= \frac{1}{T}\sum_{t=0}^{T-1} \frac{\nabla\mu(u_t|\theta, y_t)}{\mu(u_t|\theta, y_t)} \sum_{s=t+1}^{T} 0^{s-t-1} \chi_{u_{s-1}}(m^*_{s-1}) \\ &= \frac{1}{T}\sum_{t=0}^{T-1} \frac{\nabla\mu(u_t|\theta, y_t)}{\mu(u_t|\theta, y_t)} \chi_{u_t}(m^*_t) \\ &= \frac{1}{T}\sum_{t=0}^{T-1} \nabla \log \mu(u_t|\theta, y_t) \chi_{u_t}(m^*_t). \qquad (10.3)\end{aligned}$$

The gradient contribution is 0 unless $u_t = m^*_t$, so we can redefine the time index to n: the steps for which IState-GPOMDP chooses the label u_t correctly identifying m^*_t

$$\pi'(\nabla P)J_\beta = \lim_{n\to\infty} \frac{1}{T}\sum_{n=0}^{N-1} \nabla \log \mu(m^*_n|\theta, y_n),$$

which, apart from a scaling factor N/T, is a gradient estimator for batch training of the function $\mu(u_t|\theta, y_t)$ using a log-probability cost function. Richard and Lippmann [1991] show that this cost function is sufficient to produce an ANN that estimates the posterior probability $\Xi(m|\theta, y_t)$. Armed with this gradient estimator we can apply gradient ascent to train the ANN to model the posterior probabilities, which will later be converted to scaled likelihoods for use by the HMM.

This analysis shows that instead of applying IState-GPOMDP by sampling labels from $\mu(\cdot|\theta, y_t)$, we can always choose the correct label $\mu(m^*_t|\theta, y_t)$ without further biasing the gradient. This is accomplished in the step in which we change the time index from t to n. Effectively, an oracle is used to chose actions, somewhat similar to the way importance sampling can work (see Section 9.3). The analysis is also interesting because specialising the IState-GPOMDP algorithm has yielded an established machine learning method, demonstrating the generality of the POMDP model. Unfortunately, this means that our training method has not yet achieved anything more than existing approaches. The second phase of training makes use of POMDP framework features such as delayed rewards, allowing potential improvements over existing speech training methods.

10.3.1.2 Learning Transition Probabilities

We consider HMM-states to be equivalent to I-states except that HMM-state transition probabilities are not driven by observations. Thus, we use the notation $\omega(h|\phi, g)$ to refer to the probability of HMM-state transition $g \to h$.

Previously studied options for learning the state transition probabilities $\omega(h|\phi, g)$ include the popular Baum-Welch Algorithm, using an ANN to model both transitions and emissions (as in Appendix E.5.1.1), or estimating phoneme transitions directly from the data.[3] We introduce an alternative approach, using a variant of Exp-GPOMDP to estimate the phoneme transition probabilities.

The Viterbi Algorithm

At this juncture it is worth reviewing the Viterbi algorithm which is more completely described in Section D.1.2. The aim of Viterbi decoding is to find the maximum likelihood *sequence* of I-states. The algorithm can be visualised as finding the best path through an I-state lattice, depicted for 3 I-states in Figure 10.8.

The Viterbi algorithm is useful because it identifies the most likely I-state for each step given the I-state trajectory up to the current time. This equates to outputting a stream of phonemes. By comparison, the full forward probability update tells us how likely the states are assuming *all* trajectories through the I-state lattice are possible. We use the Viterbi algorithm because a speaker describes only a single trajectory through the I-state lattice. Put another way, the Viterbi algorithm specifies an I-state for each step in time.

One step of the Viterbi algorithm iterates over all I-states. For each I-state h the algorithm chooses the most likely I-state g that leads to I-state h. This is performed by evaluating an approximate form of the forward probability update (D.1)

$$\hat{\alpha}_{t+1}(h|\phi, \bar{y}_t) = \max_g \left[\hat{\alpha}_t(g|\phi, \bar{y}_{t-1})\omega(h|\phi, g)\right]\xi(y_t|h). \tag{10.4}$$

The approximation arises because of the use of the max function. We take the maximum because we want to identify the single most likely predecessor I-state. The algorithm stores the predecessor g of every I-state h for the last l time steps, giving us the ability to track the most likely sequence back from any I-state h for l steps.

To produce the optimal I-state sequence the lattice should extend back to the beginning of the sequence, however, due to memory and time restrictions, it is typical to assume that limiting the decoding to l steps introduces negligible errors [Proakis,

[3]This option is only possible when the data is labelled with all transitions. Most speech systems use HMMs with multiple states per phoneme and data that is only labelled at the phoneme level (at best), so they cannot use this simple option.

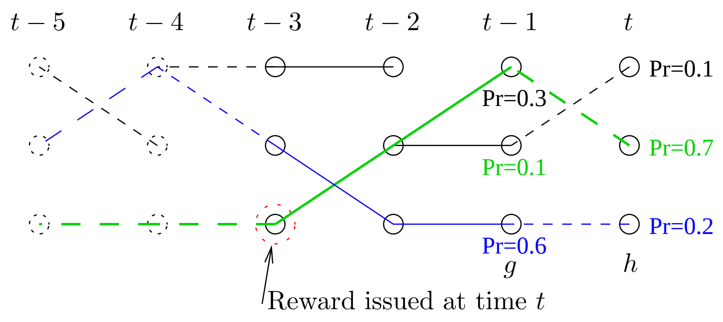


Figure 10.8: For each I-state h, select the most likely predecessor I-state g based on the probability of being in I-state g and the transition probability $\omega(h|\phi, g)$. At time $t-1$ we see that the most likely last l I-states were $\{h_{t-4} = 1, h_{t-3} = 2, h_{t-2} = 3, h_{t-1} = 3\}$ with normalised likelihood 0.6, but at time t the Viterbi procedure makes a correction that makes the last 4 most likely I-states $\{h_{t-3} = 3, h_{t-2} = 2, h_{t-1} = 1, h_t = 2\}$ with normalised likelihood 0.7.

1995, §8.2.2]. Thus, after l steps, the most likely I-state at the end of the lattice is locked in as the I-state for time step $t-l$. For example, in Figure 10.8, $l = 3$. As we update the lattice to incorporate the observation for step t, I-state 3 is locked in as the correct I-state for $t-3$ because it is at the end of the most likely sequence of I-states: the sequence $\{h_{t-3} = 3, h_{t-2} = 2, h_{t-1} = 1, h_t = 2\}$ with a normalised likelihood of 0.7.

The following sections describe how Exp-GPOMDP is used to train the speech recognition agent. We start by describing the job of the agent and what its parameters are. Then we introduce the key modification to Exp-GPOMDP that allows it to fit naturally into the context of the Viterbi algorithm for speech processing. Subsequent sections delve into the details of estimating $\nabla\eta$ using the modified Exp-GPOMDP algorithm and our chosen parameterisation. After a detailed discussion of the reward structure a dot-point summary of the training process is given.

Exp-GPOMDP using Viterbi Probabilities

For the purposes of the speech recognition agent, each I-state in the Viterbi lattice is defined to represent a single phoneme. Our speech agent is responsible for running the Viterbi decoder that emits a phoneme classification for each frame of speech. If the Viterbi lattice is length l then the agent emits the classification for frame $t-l$ at time t. As illustrated in Figure 10.8, this means the reward the agent receives at time t is a result of the classification decision for frame $t-l$. The implications of this for temporal credit assignment are discussed in Section 10.3.1.3. The agent parameters ϕ represent the phoneme transition probabilities $\omega(h|\phi, g)$. The agent is illustrated by Figure 10.9.

The task of our modified Exp-GPOMDP algorithm is to estimate the gradient of $\nabla\eta$

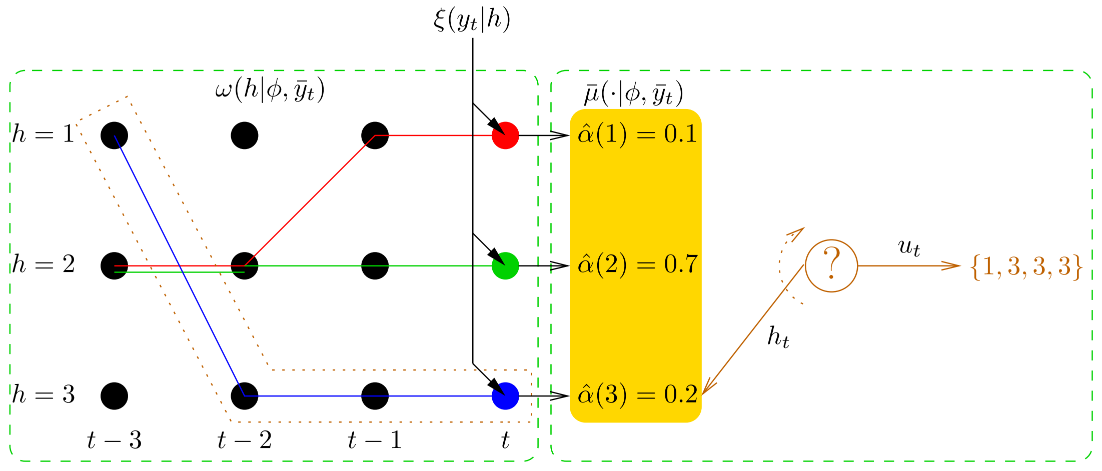


Figure 10.9: The LVCSR agent, showing the Viterbi lattice and the $\bar{\mu}(\cdot|\phi,\bar{y}_t)$ distribution from which an action is chosen. In this instance the action consists of the sequence of labels leading back through the Viterbi lattice from $h_t = 3$, that is, $u_t = \{h_{t-3} = 1, h_{t-2} = 3, h_{t-1} = 3, h_t = 3\}$.

with respect to the ϕ parameters. In other words, Exp-GPOMDP adjusts the Viterbi lattice transition probabilities to maximise the long-term average reward. The key modification to Exp-GPOMDP is a natural consequence of using a Viterbi decoder to classify speech frames: Exp-GPOMDP uses the Viterbi approximation (10.4) of the I-state belief instead of computing the full forward belief (given by Equation (6.2)). Just like the forward belief, the Viterbi approximation $\hat{\alpha}_t$ depends on ϕ, and on all past acoustic feature vectors y_t, so we write the Viterbi I-state belief for I-state (phoneme) h as $\hat{\alpha}_t(h|\phi,\bar{y}_{t-1})$.

Recall from Section 6.2 that Exp-GPOMDP does not sample I-state transitions, but it *does* sample actions. However, the Viterbi algorithm normally operates completely deterministically, choosing the most likely current I-state

$$h_t = \arg\max_h \hat{\alpha}_t(h|\phi,\bar{y}_{t-1}),$$

and tracking back l steps through the Viterbi lattice to find the best phoneme to emit as a classification action. To make our speech processing agent fit into the Exp-GPOMDP framework we must allow it to emit stochastic classification actions for frame t. This allows the agent to occasionally try an apparently sub-optimal classification that, if it results in a positive reward, will result in parameter adjustments that make the classification more likely.

Stochasticity is introduced by making the agent sample an I-state h_t from the

Viterbi phoneme probability distribution $\hat{\alpha}_{t+1}(\cdot|\phi, \bar{y}_t)$. The action conceptually consists of the entire l step sequence back through the Viterbi lattice to the optimum phoneme for frame $t - l$ *given the stochastic choice of* h_t. The reason for emitting the entire sequence instead of just the phoneme for frame $t - l$ is to ensure that all possible choices of h_t emit a unique action. The stochastic choice of h_t is forgotten during the next step and has no impact on the Viterbi probability.

Now we describe more precisely how this scheme fits into the Exp-GPOMDP algorithm. Recall from Equation (6.3) that Exp-GPOMDP evaluates the action distribution $\mu(u|\theta, h, y)$ for each I-state h, and then samples action u_t from the weighted sum of these distributions with probability $\bar{\mu}(u_t|\phi, \theta, \bar{y}_t)$. For our speech agent the action distribution for each I-state h is

$$\mu(u|h) := \chi_u(h),$$

which is a deterministic mapping from I-states to actions. This definition reflects the operation of choosing an I-state h, and from this choice, deterministically tracking back through the Viterbi lattice to find the best sequence of phonemes to emit as u.

Our choice of deterministic $\mu(\cdot|h)$ may seem strange after just stating that actions have to be chosen stochastically, however recall that Exp-GPOMDP samples the sum of weighted distributions $\bar{\mu}(\cdot|\phi, \theta, \bar{y}_t)$, which *is* stochastic, even though $\mu(\cdot|h)$ is not

$$\begin{aligned}\bar{\mu}(u|\phi, \bar{y}_t) &= \sum_{h \in \mathcal{G}} \hat{\alpha}_{t+1}(h|\phi, \bar{y}_t)\mu(u|h) \\ &= \sum_{h \in \mathcal{G}} \hat{\alpha}_{t+1}(h|\phi, \bar{y}_t)\chi_u(h).\end{aligned}$$

Thus, the probability of action u, consisting of a classification for the stochastically chosen current frame h_t plus the last l frames, is equal to the probability of frame t being classified as h_t, and is given by $\hat{\alpha}_{t+1}(h_t|\phi, \bar{y}_t)$. Thus, sampling an action from $\bar{\mu}(\cdot|\phi, \theta, \bar{y}_t)$ is equivalent to sampling an I-state h_t from the same distribution.

There are $|\mathcal{G}|^l$ total possible actions that the system can emit, but at any one time step at most $|\mathcal{G}|$ actions have non-zero probabilities because the choice of the I-state $h_t \in \mathcal{G}$ completely determines the rest of the action.

After an action has been emitted the gradient of the action is computed and used to update the Exp-GPOMDP eligibility trace z. A reward is received from the environment and it is used along with z to update the gradient estimate.

Computing the Gradient

To this point we have given a general overview of how the Exp-GPOMDP algorithm can be applied to speech. What has been left unspecified is exactly how $\omega(h|\phi, g)$ is parameterised and how to compute the gradient accordingly. There are several sensible ways of doing this. The next paragraphs outline the method used in our experiments.

We have already described the deterministic $\mu(u|h)$ function, which simplifies the problem of choosing an action u_t from $\bar{\mu}(\cdot|\phi, \bar{y}_t)$ to one of choosing an I-state, from the distribution $\hat{\alpha}_{t+1}(\cdot|\phi, \bar{y}_t)$.

What remains to describe is how to parameterise the update of $\hat{\alpha}$. In Section 6.2 we did this using Equation (6.2), which for each I-state g evaluates the probability of making a transition from I-state g to I-state h using $\omega(h|\phi, g)$, then multiplies by the current forward probability $\alpha_t(g|\phi, \bar{y}_{t-1})$. In this section we take a different approach by including the current forward probabilities in the evaluation of the soft-max function. Assuming that the step t phoneme is labelled g, and the step $t+1$ phoneme is labelled h, the likelihood of phoneme h using the Viterbi criterion is

$$\hat{\alpha}_{t+1}(h|\phi, \bar{y}_t) := \xi(y_t|h) \frac{\max_g \exp(\phi_{gh})\hat{\alpha}(g|\phi, \bar{y}_{t-1})}{\sum_{g' \in \mathcal{G}} \exp(\phi_{g'h})\hat{\alpha}(g'|\phi, \bar{y}_{t-1})}. \tag{10.5}$$

For every transition from phoneme g to phoneme h there is one parameter $\phi_{gh} \in \mathbb{R}$. The scaled likelihoods $\xi(y_t|h)$ are provided by the ANN, trained during Phase 1. In previous experiments we used the soft-max function to evaluate the probability of making a transition from a given I-state g to some I-state h, thus we normalised the distribution by summing over potential *next* I-states h' (see Section 3.4.1). When implementing the Viterbi algorithm we are given the next I-state h, so we normalise over potential *previous* I-states g'. The Viterbi algorithm dictates that the previous I-state we choose, denoted $\tilde{g}$, must be the one that maximises $\hat{\alpha}_{t+1}(h|\phi, \bar{y}_t)$.

Before deriving the gradient we introduce an approximation that greatly reduces the amount of computation needed per step of the algorithm. We make the assumption that $\hat{\alpha}_t(g)$ does not depend on ϕ. This reduces the complexity of the *gradient* calculation from being square in $|\mathcal{G}|$ to linear in $|\mathcal{G}|$. The computation of $\hat{\alpha}_{t+1}(h|\phi, \bar{y}_t)$ for all h is still square in $|\mathcal{G}|$. The approximation also allows normalisation of $\hat{\alpha}_t(g)$, which would otherwise vanish over time, without needing to further complicate the gradient computation. For comparison, we implemented the exact calculation of $\frac{\nabla\bar{\mu}}{\bar{\mu}}$ and found that with 48 phonemes/I-states, each step was about 100 times slower than using the approximation described below, with no strong evidence that it produced better results.

As discussed earlier in this section, the choice of u_t from $\bar{\mu}(\cdot|\phi, \bar{y}_t)$, is equivalent the choice of a phoneme h_t from $\hat{\alpha}_{t+1}(\cdot|\phi, \bar{y}_t)$. Thus $\frac{\nabla\bar{\mu}}{\bar{\mu}}$, required by Exp-GPOMDP, is the same as $\frac{\nabla\hat{\alpha}_{t+1}}{\hat{\alpha}_{t+1}}$ and follows the form of the soft-max derivative given by Equation (3.9). Let $\Pr(g|\phi, \bar{y}_t, h)$ be the probability of transition $g \to h$ given h and the current value of $\hat{\alpha}(g|\phi, \bar{y})$ (which is assumed to equal $\hat{\alpha}(g)$)

$$\Pr(g|\phi, \bar{y}_t, h) = \frac{\exp(\phi_{gh})\hat{\alpha}(g)}{\sum_{g' \in \mathcal{G}} \exp(\phi_{g'h})\hat{\alpha}(g')},$$

so that $\tilde{g} = \arg\max_g \Pr(g|\phi, \bar{y}_t, h)$. Then

$$\frac{\frac{\partial \bar{\mu}(u_t=h|\phi,\bar{y}_t)}{\partial \phi_{gh}}}{\bar{\mu}(u_t = h|\phi, \bar{y}_t)} = \frac{\frac{\partial \hat{\alpha}(h|\phi,\bar{y}_t)}{\partial \phi_{gh}}}{\hat{\alpha}(h|\phi, \bar{y}_t)} = \chi_{\tilde{g}}(g) - \Pr(g|\phi, \bar{y}_t, h). \tag{10.6}$$

There is no dependency on the emission probabilities $\xi(y_t|h)$ in Equation (10.6) because it vanishes when we compute the ratio $\frac{\nabla \bar{\mu}}{\bar{\mu}}$. This would not be the case if we implemented global training of the emission probabilities and transition probabilities. Then $\xi(y_t|h)$ would depend explicitly on a subset of the parameters ϕ representing the weights of the ANN that estimates $\xi(y_t|h)$. We would need to compute $\nabla^\phi \xi(y_t|\phi, h)$ by back propagating the gradient through the layers of the ANN. This idea is similar to the global ANN/HMM training scheme implemented by Bengio et al. [1992].

Reward Schedule

At each time step an action u_t is chosen from $\bar{\mu}(\cdot|\phi, \bar{y}_t)$, which according to our definition of $\bar{\mu}$ is equivalent to choosing a phoneme h_t from $\hat{\alpha}(\cdot|\phi, \bar{y}_t)$. Actions are generated by following the path back through the Viterbi lattice defined by the choice of h_t. If the Viterbi lattice is length l, then the path back gives us the previous l most likely phonemes given our choice of current phoneme h_t. The final decision for the classification of speech frame $t - l$ is the phoneme at the end of the path traced back from h_t. Because the Viterbi procedure corrects errors made at time t, we expect that there is a high probability that the correct classification is made for frame $t - l$, even when Exp-GPOMDP chooses a low probability phoneme h_t.

A reward of 1 is received for correctly identifying a *change in phoneme* at time t. A reward of -1 is received for incorrectly identifying a change in phoneme. Because the Viterbi filter delays the final classification of frames until Exp-GPOMDP has processed another l frames, the reward is delayed by l steps.

If the phoneme classification for frame t is the same as the previous phoneme classification, then no reward is issued. We only reward changes in phoneme classification because the phoneme sequence is what we really care about. Rewarding the accuracy of every frame would cause the system to try and accurately match the length of each phoneme as well as the sequence of phonemes, hence wasting training effort. For example, suppose the true frame-by-frame labels are {`b, b, eh, eh, eh, d, d`} and our Exp-GPOMDP trained classifier emits {`b, eh, eh, eh, d, d, d`}; then both sequences are interpreted as the sequence of phonemes `b-eh-d`, and our classifier should be given the maximum reward. However, if we were to base rewards on the frame-by-frame labels then Exp-GPOMDP would receive penalties for the apparent mistakes at the 2nd and 5th frames. The focusing of training effort to maximise an arbitrary high-level criteria is one of the key benefits of the POMDP model. HMM training would penalise the 2nd and 5th frames by interpreting the emission of those phonemes as zero probability events.[4] Off-by-one frame errors such as we have described are fairly common because the hand labelling of the data set is only accurate to a few ms.

Furthermore, penalties are received *only for errors not corrected by Viterbi decoding.* The system is free to artificially raise or lower transition probabilities from their maximum likelihood values, provided doing so decreases errors after Viterbi decoding. Training automatically adjusts for the ability of the Viterbi decoder to correct errors. We do not rely on Viterbi decoding as a post-training method for reducing errors.

The final operation of the system is the same as an ANN/HMM hybrid. The training procedure is not the same because we avoid using the maximum likelihood criterion.

[4]This assumes that the HMM has been constrained to exactly match the hand-labelled data.

Summary of Transition Probability Training

Combining all the elements of this section we end up with the following training procedure for each frame of input speech:

1. A frame of the speech waveform is pre-processed to form a vector of features such as power and mel-cepstral values. These features form the observation vector y_t for the t'th frame of speech.

2. We feed y_t forward through the ANN, trained during Phase 1, which outputs an estimate of $\Xi(m|y_t)$ for each phoneme m.

3. $\Xi(m|y_t)$ is converted into a scaled likelihood $\xi(y_t|m)$ by dividing the posterior probabilities by the phoneme probabilities, which are estimated from the training set.

4. For each phoneme, represented by an I-state h in the Viterbi lattice, we evaluate $\hat{\alpha}_{t+1}(h|\phi, \bar{y}_t)$ using Equation (10.5).

5. A path back through the Viterbi lattice is chosen by sampling a phoneme h_t from $\hat{\alpha}_{t+1}(\cdot|\phi, \bar{y}_t)$.

6. We compute $\frac{\nabla^{\phi}\bar{\mu}}{\bar{\mu}}$ for the chosen path by evaluating Equation (10.6). This log gradient vector is added to the Exp-GPOMDP trace z_t.

7. Follow the path back through the Viterbi lattice from phoneme h_t to construct the action u_t. Emit the phoneme at the end of the Viterbi lattice as the correct phoneme for frame $t-l$.

8. Issue $r_{t+1} = 1$ if the phoneme for $t-l$ correctly identifies a change in phoneme, $r_{t+1} = 0$ for no change, and $r_{t+1} = -1$ for an incorrect (or missing) change in phoneme. Increment t, goto step 1.

10.3.1.3 Eligibility Trace Filtering for LVCSR

This section describes the design of eligibility trace filters, as described in Section 9.1, used to improve the performance of the LVCSR experiments. We present 5 heuristic filters for temporal credit assignment in the LVCSR domain. Section 10.3.2.3 is an experimental comparison of the filters, resulting in the selection of a filter for the larger experiments.

Rewards are a function of the I-state transitions that were followed an indefinite number of steps into the past. This dependency arises through the dependence of the reward on the Viterbi probabilities. The more peaked the Viterbi probabilities are around the correct phoneme, the more likely the agent is to receive a positive reward. The effect of past transitions on the current Viterbi probabilities become progressively smaller as time goes on, possibly at an exponential rate. For example, humans need to recall a couple of words of context to achieve good recognition, but we rarely need to remember what was said a minute ago just to make out the words currently being spoken. Thus, credit may need to be assigned to I-state transitions followed indefinitely far into the past, but less credit should be assigned to old transitions. The standard *exponential* discounting can model this situation efficiently. We experiment with four versions of this filter corresponding to $\beta = 0.3$, 0.6, 0.9, and 0.99. The exponentially discounted case is the first filter shown in Figure 10.10, aligned with a Viterbi lattice of length $l = 3$. We also experiment with a *linear* decaying filter that assumes that the effect of transitions is negligible after $2l$ steps.

We should be able to do better by taking into account the length of the Viterbi lattice. The length of the Viterbi lattice does *not* affect the Viterbi probabilities, but it does affect the rewards because the reward is based on phoneme transitions chosen by Viterbi decoding. If the Viterbi lattice has length l, then r_t is based on the classification of the frame for time $t - l$, which depends on one of the transitions followed from step $t - l - 1$ to $t - l$. High credit for r_t should be given to gradient component computed at time $t - l$, that is, the gradient component with a delay of $\tau = l$. However, the transitions between $t - l$ to t, and the decision about which lattice path to emit at time t, also affect the reward, so these transitions cannot be ignored in the credit assignment.

This reasoning motivates the next two filters shown in Figure 10.10. The first is a *triangular* filter, assigning maximum credit to the transition between $t - l - 1$ and $t - l$; the next looks like a *ramp*, making no assumptions about how to assign credit between time t and $t - l$. Both have a linear decay in credit assignment after $t - l$, reflecting the decaying impact of transitions on $\hat{\alpha}(\cdot|\phi, \bar{y})$ as they become older. The more coefficients the filter has, the slower it is to compute, thus we decay the impulse response quickly, using a maximum of 9 filter taps. This also limits the length of the Viterbi lattice since we must model its effects with a limited number of filter co-efficients. We experimented with up to 9 coefficients and a Viterbi lattice of up to 4 steps.[5]

The final filter we define makes no assumptions about the credit assignment except that no assignment should take place for delays greater than $\tau = 2l$, which implies a *rectangular* filter. The results of applying each of these filters are shown in Section 10.3.2.3. The ramp filter performed best and was used for the main transition training experiment outlined in Section 10.3.2.5.

[5]We have avoided IIR filters (except for the exponentially discounted filters) because they generally seem to increase the variance of estimates.

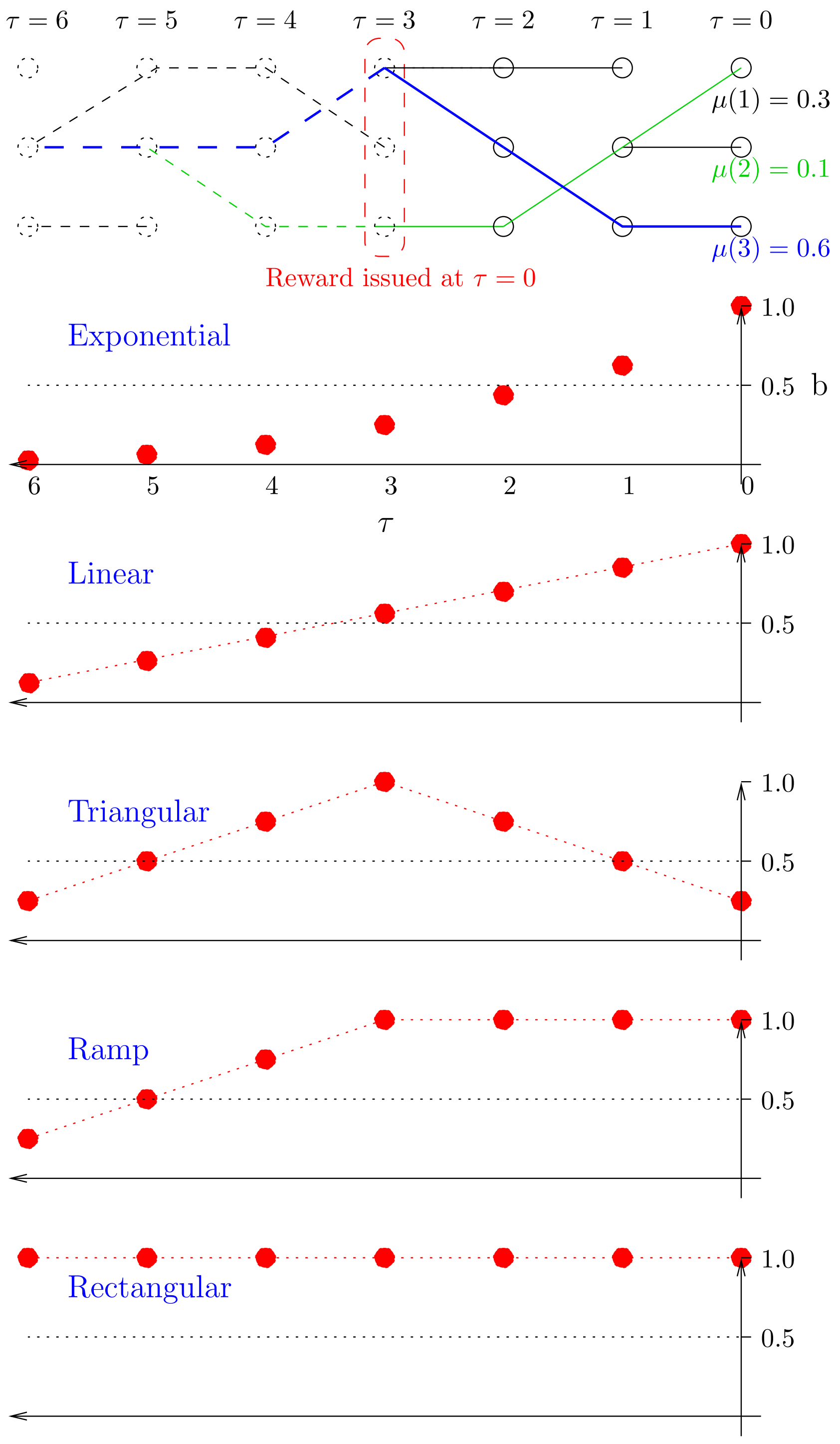


Figure 10.10: The trace filters applied to the LVCSR scenario. All the filters are shown aligned with the Viterbi lattice of length $l = 3$. To achieve alignment we reversed the time axis of the impulse responses so that τ increases to the left. The b axis of the filters gives the value of the b filter parameter for each τ. For all filters $a = [1]$.

10.3.2 LVCSR Experiments

This section describes experiments that implement the methods described in the previous sections.

10.3.2.1 Emission Training Protocol

This section describes the settings used in the first phase of training, described in Section 10.3.1.1. The training set was the male portion of the TIMIT continuous speech corpus [Garofolo et al., 1993]. We used all the SI and SX sentences for the 326 male speakers in the recommended training set. This gives a total of 2608 sentences. The test set contains another 112 speakers for a total of 896 sentences.

The TIMIT continuous speech database is labelled with 63 different phonemes. Some of these phonemes are allophones, or groups of phonemes that sound the same and rely on context to distinguish them. For low level phoneme classifiers it is usual to combine groups of allophones into one phoneme, eliminating the ambiguity that they represent. The distinction can be re-introduced by higher level language modelling. We mapped the 63 TIMIT phonemes to 48 phonemes using the mapping given in Lee [1989] and reproduced in Appendix F.3.

Acoustic vectors were calculated every 10 ms using a 25 ms Hamming window, providing a total of 786,822 training frames (7868 s of speech) and 269,332 test frames (2693 s). We used the same features as the digit discrimination experiment of Section 10.2.3.1: 12 mel-cepstral parameters with a pre-emphasis factor of 0.97, plus log energy, delta and double-delta features, for a total observation dimension of 39. The cepstral features were shifted to have zero-mean. Again, features were computed off-line using the `HTK` V3.0 toolkit [Woodland, 2001]. We made no attempt to optimise them for our setting. All features were normalised to lie between $[-1, 1]$ for presentation to the ANN.

At each time step the observation y_t was 9 pre-processed speech frames, the current frame plus the four past frames and four future frames, spanning 90 ms of speech. The past and future context provides the ANN with a small window of history to help the classification, similar to methods such as Window-Q (see Section 2.6.2.2), but with access to 4 future acoustic vectors as well as 4 past vectors.

The network had 352 inputs, 1000 hidden nodes and 48 outputs. The hidden layer was squashed using the tanh function. This architecture was chosen because it was used by Bourlard and Morgan [1994]. Training was greatly speeded up by using the 4-way parallelism provided by the single-instruction multiple-data instruction set of the Pentium III processor (see Chapter 11).

The memory-less IState-GPOMDP parameters were chosen to replicate the training of an ANN that maximises the log-likelihood of correct model, as described in Section 10.3.1.1. We used discount factor $\beta = 0$, gradient estimation time $T = 786,822$ (the number of training frames), and quadratic penalty $\wp = 0.05$. Recall from Equation (10.3) that we can allow the agent to always choose the correct action $u_t = m_t^*$. Thus, the estimation time of $T = 786,822$ is simply one pass through the training data, as if we were batch training an ANN. The weights were initialised to values between $[-0.01, 0.01]$. We used cross-validation on the test set, but even with 401,000 parameters we did not observe any over-fitting.

10.3.2.2 Emission Training Results

We performed one long training run that took approximately 48 hours and 240 gradient ascent iterations using 96 Pentium-III 550 MHz CPUs from our “Bunyip” Beowulf cluster.

Recall that the ANN estimates the posterior probability $\Xi(m|y_t)$ for each phoneme m. The maximum posterior criterion, which chooses

$$m_t = \arg\max_m \Xi(m|y_t),$$

resulted in 62.1% of 269,332 frames being correctly classified.

Further testing used a Viterbi decoding procedure with $l = 3$ and a single I-state in the Viterbi lattice per phoneme. The likelihood of an observation given a phoneme, as used by the Viterbi algorithm, was

$$\xi(y_t|m) = \frac{\Xi(m|y_t)}{\Pr(m)},$$

where $\Pr(m)$ is the prior probability of phoneme m, estimated by counting the number of 10 ms frames labelled with each phoneme. Then we estimated the *transition* probabilities by counting phoneme transitions in the training data. Applying Viterbi decoding using the estimated transition probabilities improved the results to 68.6%. When the data is fully labelled with phonemes, and we use 1 I-state per phoneme, counting the transitions gives the same I-state transition probabilities that the Baum-Welch algorithm would estimate. Thus, 68.6% is a baseline result showing what could be achieved using maximum likelihood training.

10.3.2.3 Trace Filtering Protocol

The aim of this experiment was to compare the eligibility trace filters described in Section 10.3.1.3. This allowed us to select the filter with the best average performance for performing the main transition probability training experiment. We applied the

Table 10.2: The long-term reward η achieved by each of the 8 trace filters described in Section 10.3.1.3. Results are averaged over 30 runs. Values for η are multiplied by 10^2. The variance is scaled to match.

Filter	*mean* η	*max.* η	*min.* η	*var.*
$\beta = 0.3$	7.00	7.09	6.99	5.18×10^{-4}
$\beta = 0.6$	8.22	10.2	6.95	1.15
$\beta = 0.9$	7.77	8.16	7.23	0.597
$\beta = 0.99$	8.00	8.09	7.80	0.0601
Linear	8.773	10.1	6.81	0.695
Triangular	7.61	9.01	6.70	0.699
Ramp	**8.85**	**10.0**	**6.95**	**0.440**
Rectangular	7.04	9.43	6.70	0.542

transition training procedure (see Section 10.3.1.2) using each of the filter candidates shown in Figure 10.10.

We used a vastly reduced training set of 705 speech frames over 2 sentences.[6] This speeded up training, allowing a reasonable number of trials to be conducted. The test data set was the same as the training set.

Using trial and error we chose a gradient estimation time of $T = 4 \times 705$, being the smallest multiple of the number of frames for which the gradient estimates were consistent. No quadratic penalty was applied. Transition parameters were initialised to [0]. Recall that the transition parameters are real values that are converted into probabilities through the soft-max function, so initialising parameters to [0] assumes uniform transition probabilities. This does not create the zero-gradient problems described in Chapter 7 because the transitions are also driven by the ANN emission probabilities. The emission probabilities were provided by the ANN trained on the entire training set.

10.3.2.4 Trace Filtering Results

The best results were obtained with a Viterbi filter of length $l = 3$ and a maximum of 7 filter co-efficients. The long-term average rewards obtained using these parameters are shown in Table 10.2. The results are averaged over 30 runs.

The best run, scoring $\eta = 0.102$, using $\beta = 0.6$, achieved a frame-by-frame accuracy of 94.6%. In contrast, the result from the maximum likelihood estimation of the transition probabilities, determined by counting the transitions in the 705 frame training set, was 93.2%. The 20% relative error reduction achieved by the POMDP

[6]Specifically, we used the TIMIT training sentences si1616 and sx401 from the first male speaker.

approach is an interesting result. It indicates that the use of a reward function that maximises what we care about: the average accuracy at the frame level, can result in better performance than the maximum likelihood training criterion.

Because this result is based on the *maximum* long-term average η, we cannot claim that the result is significantly better than ML in a statistical sense. However, because the maximum likelihood procedure for choosing weights has 0 variance, any η above the ML result must be due to the potential benefits of POMDP training methods over ML training methods.

The Viterbi filter length of $l = 3$ is very short, providing error correction only up to the $t - 3$'th frame. We changed the filter length to $l = 20$ for testing purposes, still using the transition parameters trained with $l = 3$. The frame accuracy increased to 94.9%, a 0.3% improvement over $l = 3$. Given that the accuracy with no Viterbi filter ($l = 0$) was 91.2%, we see that the majority of errors were corrected with a filter of length $l = 3$. *Training* with $l = 20$ might improve results further, although temporal credit assignment becomes harder as l increases. The rest of the results in this chapter assume $l = 3$.

We selected the ramp filter for the full training set experiment because it has the best *mean* result. It is worth mentioning that a single sided t-test reveals that the ramp filter is *not* statistically better than the linear filter for only 30 trials.

The accuracy of the sequence of phonemes is more important than the frame-by-frame accuracy. This is reflected by our choice of reward function that only rewards and penalises *changes* in phoneme classification. The phoneme accuracy usually appears poorer than the frame-by-frame accuracy because the majority of phonemes are accurately classified, and phonemes last for many frames. Over the 705 frames of our training set there are 83 phonemes. The Exp-GPOMDP trained Viterbi transitions resulted in a phoneme classification accuracy of 90.36%.[7] Viterbi transitions estimated using counting resulted in an accuracy of 89.1%.

10.3.2.5 Transition Probability Training Protocol

The training procedure is described in Section 10.3.1.2. We tried both 1 and 2 I-states per phoneme, for a total of $|\mathcal{G}| = 48$ and $|\mathcal{G}| = 96$ respectively. Tied observation distributions were used in the latter case so that both I-states for a particular phoneme used the same ANN trained value for $\xi(y_t|m)$.

We used the full data set and training set as described for the emission training experiment of Section 10.3.2.1. The gradient estimation time was $T = 37,767,456$,

[7]Phoneme accuracy is measured using $\frac{\text{phonemes}-(\text{insertions}+\text{additions}+\text{substitutions})}{\text{phonemes}}$, which is a standard measure used by Lee [1989] and Bengio et al. [1992]. The number of insertions, additions and substitutions was measured using `GNU diff` v2.7.2.

being the number of phonemes multiplied by the number of training samples. This was chosen on the heuristic basis that it allowed the system to see the data set at least as many times as there are phonemes, allowing Exp-GPOMDP a reasonable chance to explore many different choices of u_t for each observation.

To reduce training time the quadratic penalty was not used. Further speed-up was achieved by using a traditional line search during gradient ascent, searching for peaks in the *values* as the step size was increased. This is in contrast to the majority of the experiments in thesis which used the GSEARCH algorithm (described in Appendix B.1). A further reason to prefer the use of the value line search is that the GSEARCH algorithm appeared to often over-step the true maximum η along the search direction. This may be a consequence of the approximation discussed in the "Computing the Gradient" section. The transition weight parameters ϕ were initialised to values between $[-0.05, 0.05]$.

10.3.2.6 Transition Probability Training Results

Training with 1 I-state per phoneme resulted in only a fractional accuracy increase compared to the optimum maximum likelihood weights. On the test set of 269,332 frames, 8 previously incorrectly classified frames became correctly recognised. However, the fact that there was any improvement is significant.

Our best run used 2 I-states per phoneme. It took approximately 2 days using 96 Pentium III 550 MHz CPUs from our "Bunyip" Beowulf cluster. The final frame-by-frame accuracy is 69.4%. This is a 7.3% improvement from the 62.1% result, which is the accuracy based on the output $\Xi(m|y_t)$ of the phase 1 ANN. Exp-GPOMDP adjusted the I-state transitions to reinforce likely phoneme sequences. It is not producing classifications based only on the most likely instantaneous phoneme probabilities from the ANN.

Overall, this is a 0.8% improvement over the optimum ML solution which counted transitions in the data. Although a small increase, the result shows Exp-GPOMDPs ability to estimate gradients for a complex function, incorporating internal state to perform a difficult task in a POMDP framework. It also demonstrates potential benefits over ML training.

In terms of *phoneme* sequence accuracy, the trained system achieved 61.1% on the test data of 33,097 phoneme transitions. The breakdown of errors is 2174 deletions, 6524 additions, and 6150 substitutions. The optimum ML decoder achieved an accuracy of 59.2% with a similar breakdown of errors.

The accuracies quoted for this experiment are much lower than the accuracies quoted for the trace filter experiment. There are two reasons for this: (1) the much smaller training set used for the filter experiments did not contain examples of all the phonemes, greatly simplifying the problem by allowing the agent to turn off transitions to those phonemes; (2) the filter experiment used the same data for training and testing, so over-fitting occurred due to the small number of samples.

Our architecture is based on that designed by Bourlard and Morgan [1994]. They achieved a frame-by-frame accuracy of 54.8% on an 152 speaker subset of the TIMIT database with 64 phonemes. Since we used almost identical acoustic features and ANN, our better results (69.4%) are probably attributable to our larger training set and smaller set of phonemes. The highest accuracy we are aware of for the TIMIT database is 86% *phoneme* classification by Bengio et al. [1992]. This system used sophisticated tri-phone HMM models and an ANN to incorporate hand designed features for hard to classify phonemes.

10.3.3 Discussion

It is worth emphasising that this chapter is a demonstration application of policy-gradient methods to a difficult real-world problem. We do not claim that our procedures are the best way to process speech. There are several clear deficiencies with the approach we have outlined.

- Ideally, we would run both training phases simultaneously, globally optimising the performance and taking full advantage of the generality of the POMDP model. However, the system would be considerably more complex and hence slower to train.
- Enforcing tied distributions for all I-states that represent phoneme m is a poor assumption that could be rectified by global training.
- We need to better investigate the consequences of the gradient approximation that assumes the current Viterbi I-state belief is independent of the parameters ϕ. Unfortunately, it is very time consuming to do such comparisons for reasonable estimation lengths.
- By using an existing speech classifier architecture we adopt the deficiencies of that model.
- Using Monte-Carlo POMDP methods for speech has the disadvantage of being slow compared to HMM algorithms.
- This work studied frame-by-frame and phoneme classification only. Language modelling is a significant (and complex) aspect of real speech processing systems that would improve our low-level results.

10.4 Summary

Key Points

- I Signal classification can be cast as a POMDP.
- II A variant of Exp-GPOMDP that uses the Viterbi algorithm to track I-state beliefs can be used to perform phoneme level LVCSR.
- III The IIR trace filtering method of Section 9.1 can be applied to real world problems, resulting in improved average rewards and robustness through lower variance.
- IV The generality of the POMDP framework offers advantages over existing speech training methods. One example is the use of delayed rewards to avoid penalising classifier errors that do not impact on the high-level performance.
- V We have demonstrated the use of policy-gradient methods on an large-scale real-world application.

Further Work

Speech processing is such a large and complex field that is difficult to do anything but scratch the surface in one chapter of a thesis. There is need for more basic research to establish if using POMDP methods can compete with state-of-the-art speech tools. It is difficult to accurately compare results with other speech systems due to differences between data sets, the pre-processing of the speech wave-forms, and the accuracy metrics. The best way to ensure fair comparisons will be to build a pure HMM based system in parallel, using the same data set, phoneme set, and acoustic features.

In the LVCSR experiments we constrained ourselves to using an architecture for $\omega(h|\phi, g, y)$ and $\mu(u|\theta, h, y)$ that is equivalent to an existing model of speech. Other architectures should be investigated with the aim of avoiding the known flaws of existing speech models. For example, we have already experimented with training a separate neural network for each I-state. The idea is that I-state transitions represent long-term dependencies in the speech signal and the network for each I-state provides the appropriate local frame classification given the long-term state. This model is much more general that the one we finally used, but required hundreds of thousands of parameters to be estimated, causing long training times and poor performance.

In this chapter we assumed an intermediate level reward scheme that works at the phoneme level. We need to explore the effects of using higher level rewards, for example, the sentence level. As the reward becomes more sparse and abstract, the temporal credit assignment problem becomes harder.

Chapter 11

Large, Cheap Clusters: The 2001 Gordon-Bell Prize

A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.

—Leslie Lamport

The algorithms we have described can take up to 2 days to run for scenarios such as Heaven/Hell and LVCSR. They would have taken longer but for work spent designing the "Bunyip" Linux cluster and the implementation of efficient and distributed artificial neural network training software.

Some of the methods we developed carry over into other fields. For example, fast matrix-matrix multiplication and fast cluster communication methods. We place significant emphasis on the cost-effectiveness of our solution. In particular we achieved a sub USD $1 per MFlop per second price/performance ratio that was recognised with an international Gordon Bell award in November 2000.

The work of this Chapter was a team effort. In particular, the authors of the original paper [Aberdeen et al., 2000] were myself, Jonathan Baxter, and Robert Edwards.[1] Jonathan Baxter and Robert Edwards designed and installed the hardware. Jonathan Baxter also wrote the early version of the Japanese OCR code. I wrote the optimised code including the fast matrix-matrix multiplication, fast reduce, and fine tuning of the neural network communication code.

11.1 Ultra Large Scale Neural Networks

ANNs have found wide-spread use in many domains, some we have seen already, others include: character recognition, signal processing, medical diagnosis, language modelling, information retrieval, and finance prediction. A typical network in such an

[1]Jonathan Baxter was my advisor at the Research School of Information Science and Engineering at the Australian National University. Robert Edwards worked at the Department of Computer Science. The project was supported by the Australian Research Council, an Australian National University major equipment grant, and LinuxCare Australia. Thanks also to the people who made valuable contributions to the establishment and installation of Bunyip: Peter Christen, Chris Johnson, John Lloyd, Paul McKerras, Peter Strazdins and Andrew Tridgell.

application has 100–100,000 adjustable parameters and requires a similar number of training patterns in order to generalise well to unseen test data. Provided sufficient training data is available, the accuracy of the network is limited only by its representational power, which in turn is essentially proportional to the number of adjustable parameters [Anthony and Bartlett, 1999, Thm. 8.9]. Thus, in domains where large volumes of data can be collected — such as speech, face and character recognition, and web page classification — improved accuracy can often be obtained by training networks with greater than 100,000 parameters. For example, Section 10.3.2.1 described how we trained a large ANN with 401,000 parameters and 786,822 training vectors without observing over-fitting.

This chapter describes a distributed method for training *ultra large scale neural networks* (ULSNNs), or networks with more than one million adjustable parameters and a similar number of training examples. At its core, the algorithm uses Emmerald: a single-precision (32 bit) general matrix-matrix multiply (SGEMM) based on the Pentium III SIMD Streaming Extensions (SSE). Emmerald has a peak performance in excess of 1090 MFlops/s on a single 550 MHz Pentium III. Empirically we found single precision sufficient for gradient-based training of ULSNN's. For medium to large scale ANNs as few as 16 bits precision is sufficient [Asanović and Morgan, 1991].

To illustrate the use of large and cheap clusters for ULSNN training we focus on an experiment in which a neural network with 1.73 million adjustable parameters is trained to recognise machine-printed Japanese characters. The data set contains 9 million training patterns. Previous chapters have briefly described the "Bunyip" cluster on which the large experiments in this thesis were run. The following section describes it in detail. Section 11.3 describes the fast low-level matrix-matrix multiplication. Section 11.4 describes the distributed ULSNN training procedure. Finally, Section 11.6 describes the results for the Japanese OCR problem.

11.2 "Bunyip"

Bunyip is a 98-node, dual Pentium III Beowulf-class system running (upon inception) Linux kernel 2.2.14. The main design goals for this machine were to maximise CPU and network performance for the given budget of AUD $250,000 (USD $149,125 at the time of purchase). Design decisions and construction were carried out by Jonathan Baxter and Robert Edwards. Figures are quoted in US dollars for the remainder of this chapter.

11.2.1 Hardware Details

The Intel Pentium III processors were chosen over Alpha or SPARC processors for price/performance reasons. Dual-CPU systems were preferable as overall cost and size per CPU is lower than single-CPU or quad-CPU systems. Intel PIII, 550 MHz CPUs were eventually selected as having the best price/performance available at that time, taking into consideration our desire to use the floating-point single instruction, multiple data facilities of the PIII chips. AMD Athlon and Motorola/IBM G4 systems also have these facilities but were not available in dual-CPU configurations.

Off-the-shelf components were used for the networking. Gigabit ethernet was considered, but deemed too expensive at around $300 per node for the network interface card (NIC) and around $1800 per node for the switch. Instead, a novel arrangement of multiple 100 Mb/s NICs was selected with each node having three NICs that contributed some $65 per node (plus switch costs listed below).

The configuration for each node is two Pentium III 550 MHz CPUs on an EPoX KP6-BS motherboard with 384 MBytes RAM, 13 GByte UDMA66 (IDE) hard disk, and three DEC Tulip compatible 100 Mb/s network interfaces, one of which has Wake-On-LAN capability and provision for a boot ROM. The nodes have no removable media, no video capability and no keyboards. Each node cost $1282.

With reference to figure 11.1, the 96 nodes are connected in four groups of 24 nodes arranged as a tetrahedron with a group of nodes at each vertex. Each node in a vertex has its three NICs assigned to one of the three edges emanating from the vertex. Each pair of vertices is connected by a 48-port Hewlett-Packard Procurve 4000 switch (24 ports connecting each way). The switching capacity of the Procurve switches is 3.8 Gb/s. The bi-sectional bandwidth of this arrangement can be determined by looking at the bandwidth between two groups of nodes and the other two groups through 4 switches, giving a total of 15.2 Gb/s. The 48-port switches cost $2386 each.

Two server machines, identical to the nodes with the addition of CD-ROM drives, video cards, and keyboards, are each connected to a Netgear 4-port Gigabit switch that is in turn connected to two of the HP Procurve switches using gigabit links. The two server machines also act as connections to the external network. Two hot-spare nodes were also purchased and used for development and diagnostic work when not required as replacements for broken nodes.

11.2.2 Total Cost

All up we spent 98 × $1282 ($125,636) on the computational nodes (including the two hot-spares), $17,594 on the six 48-port and the 4-port gigabit switches (6 × $2386, 2 × $894 (gigabit interfaces) and $1490 for the gigabit switch), $3870 on servers (including

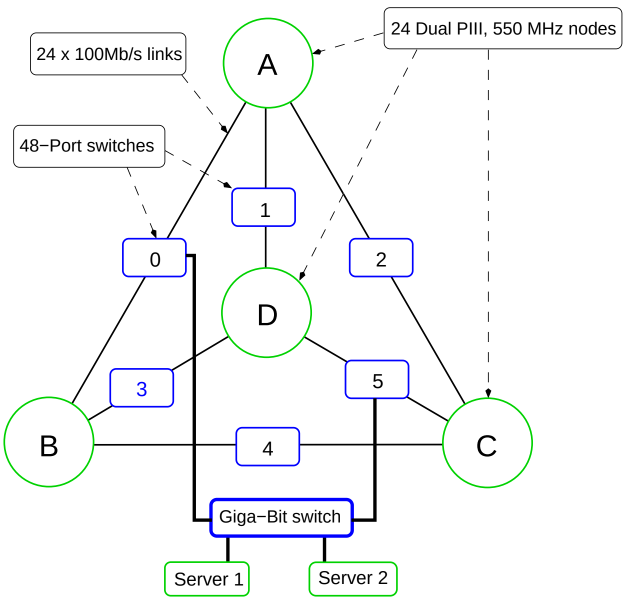


Figure 11.1: Bunyip architecture

gigabit NICs, monitors etc.), \$944 for network cables, \$179 on electrical work, \$238 on power cables and power boards, and \$298 on boot EPROMs. The ex-library shelving was loaned to us, but would have cost \$354 from a local second-hand furniture shop. Although no component was explicitly budgeted for staff time, this amounted to about 3 weeks to assemble and configure the machine which adds approximately \$1800 to the overall cost of the machine. All up, the total cost was USD \$150,913.

11.3 Emmerald: An SGEMM for Pentium III Processors

This section introduces Emmerald, the highly optimised software kernel of our UL-SNN training system. It provides a single-precision, dense, matrix-matrix multiplication (SGEMM) routine that uses the single instruction, multiple data (SIMD) features of Intel PIII chips SIMD Streaming Extensions (SSE). The SSE provide a set of new floating-point assembler instructions that operate simultaneously on four single-precision floating-point numbers.[2] Emmerald outperforms a naive (3-loop) matrix-matrix multiply by 8 times for square matrices of size 64, and a peak of 29 times for matrices of size 672. Emmerald can be downloaded from `http://csl.anu.edu.au/~daa/research.html`.

[2]Since the original work in 1999 the Pentium IV and other recent CPUs have introduced double-precision SIMD instructions, which will not be considered.

```
for (x = 0; x < m; x++) {                              /* outer loop */
    for (y = 0; y < n; y++) {
        for (z = 0; z < k; z++) {                      /* inner loop */
            c[x*n + y] += a[x*k + z]*b[z*n + y];
        }
    }
}
```

Figure 11.2: Naive matrix multiply code showing the outer loop and the inner loop.

11.3.1 Single Precision General Matrix-Matrix Multiply (SGEMM)

Within our knowledge, the matrix-matrix multiplication algorithm with the least computational complexity is Strassen's algorithm [Strassen, 1969], requiring approximately $O(n^{2.81})$ floating-point operations to multiply 2 square matrices each with $n \times n$ elements. The naive 3-loop algorithm shown in Figure 11.2 requires $O(n^3)$ operations. Unfortunately, Strassen's Algorithm is difficult to implement efficiently because the memory access patterns of Strassen's algorithm tend to prevent the efficient use of processor caches [Thottethodi et al., 1998].

Our approach has the same computational complexity as the naive algorithm. Although this complexity is fixed, skillful use of the memory hierarchy dramatically reduces overheads not directly associated with floating-point operations. Memory hierarchy optimisation combined with the use of SIMD instructions give Emmerald its performance advantage.

Emmerald implements the `sgemm` interface of Level-3 BLAS, and so may be used to improve the performance of single-precision libraries based on BLAS (such as LAPACK [Dongarra et al., 1990]). The BLAS GEMM computes

$$C \leftarrow \alpha op(A)op(B) + \beta C,$$

where $op(A)$ optionally transposes A and α, $\beta \in \mathbb{R}$ are arbitrary scalars. Multiplication of AB requires $2mnk$ floating-point operations where A is an $m \times k$ matrix and B is a $k \times n$ matrix.

There have been several attempts at automatic optimisation of GEMM for deep-memory hierarchy machines, most notable are PHiPAC [Bilmes et al., 1996] and `ATLAS` [Whaley and Dongarra, 1997]. `ATLAS` in particular achieves performance close to commercial vendor optimised GEMMs. At the time of this research neither `ATLAS` nor PhiPAC made use of the SIMD instructions on the PIII for their implementations of `sgemm`. `ATLAS` has since incorporated SIMD instructions, partly based on Emmerald and then further optimised and extended to double precision.

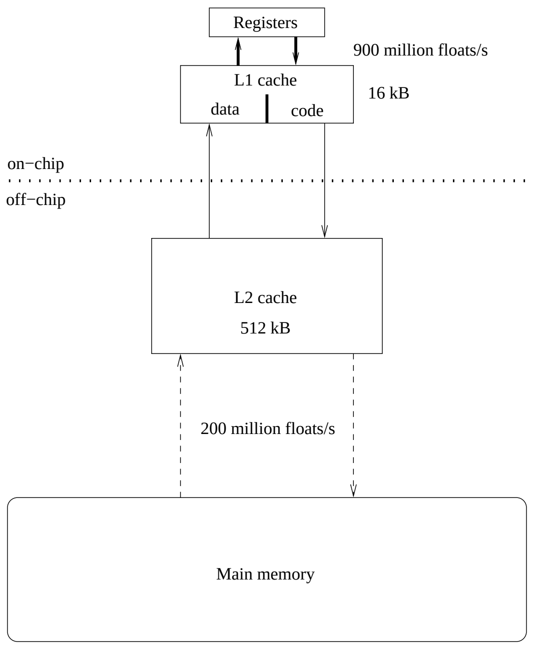


Figure 11.3: The deep memory hierarchy of the Intel PIII chip. Most current desktops vary only in the number of registers and the cache sizes.

11.3.2 SIMD Parallelisation

Modern computers have 4 levels of memory, shown in Figure 11.3. Registers are fast but small and main memory is slow but large. Caches sit between the two, becoming successively slower and larger.

A SIMD GEMM must aim to minimise the ratio of memory accesses to floating-point operations. We employed two core strategies to achieve this:

- accumulate results in registers for as long as possible to reduce write backs to cache;
- re-use values in registers as much as possible.

There are 8 SIMD (128 bit) registers available, each holding 4 single precision (32 bit) floating-point numbers. Greer and Henry [1997] computed several dot-products in parallel inside the innermost loop of the GEMM. Taking the same approach we found experimentally that 5 dot-products in the inner loop gave the best performance. Figure 11.4 shows how these 5 dot products use SIMD parallelism.

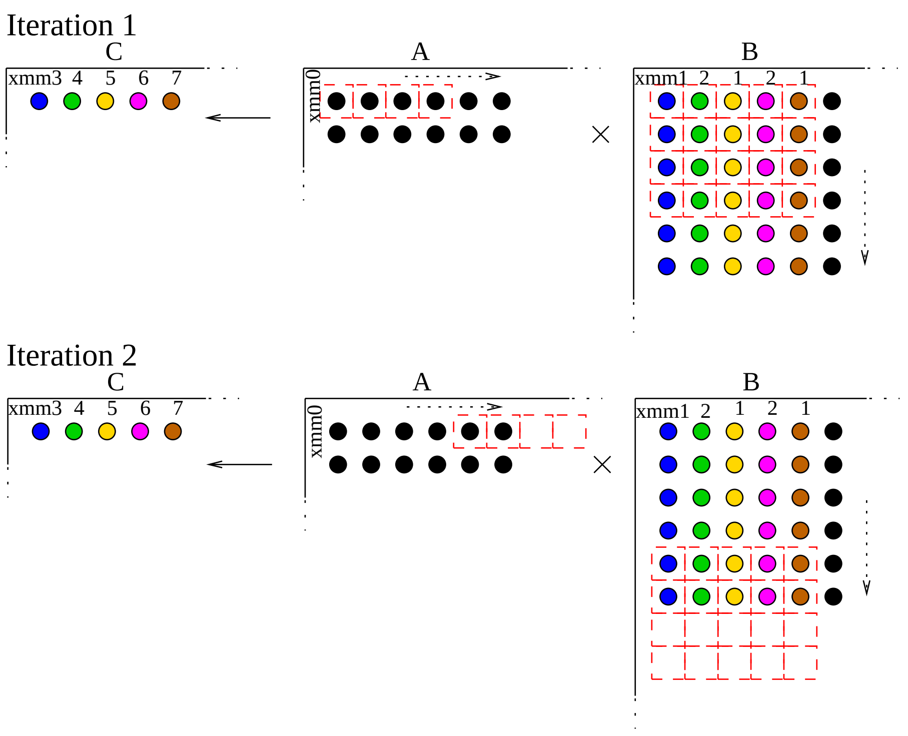


Figure 11.4: Allocation of SSE registers (labelled as `xmm[0-7]`), showing the progression of the dot products that form the innermost loop of the algorithm. Each black circle represents an element in the matrix. Each dashed square represents one floating-point value in an SSE register. Thus four dotted squares together form one 128-bit SSE register. Results are accumulated into the first 5 elements of the C matrix.

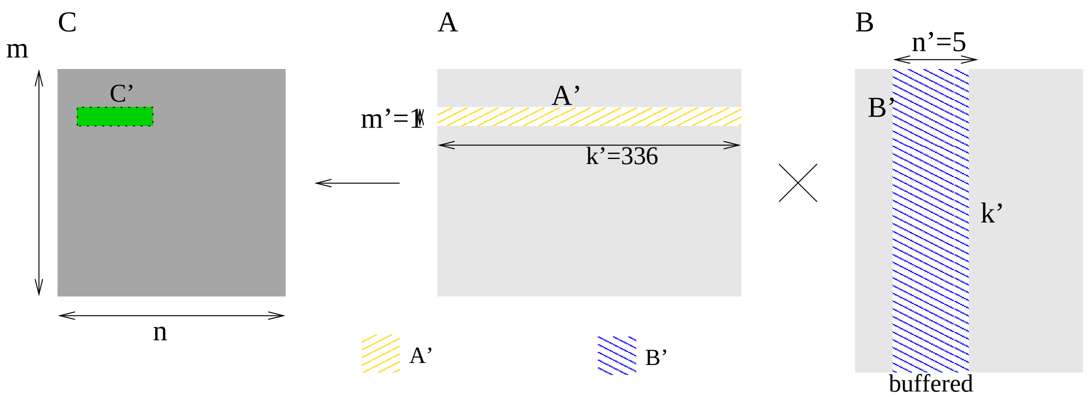


Figure 11.5: L1 blocking for Emmerald: $C' \leftarrow A'B'$ where A' and B' are in L1 and C' is accumulated in registers.

11.3.3 Optimisations

A number of other standard methods are used in Emmerald to improve performance. The details can be found in Aberdeen and Baxter [2001]. Briefly, they include:

- *L1 blocking*: Emmerald uses matrix blocking [Greer and Henry, 1997, Bilmes et al., 1996, Whaley and Dongarra, 1997] to ensure the inner loop is operating on data in L1 cache. Figure 11.5 shows the L1 blocking scheme. The block dimensions m' and n' are determined by the configuration of dot-products in the inner loop (see Section 11.3.2) and k' was chosen so that the B' block just fits in L1 cache.
- *Unrolling*: The innermost loop is completely unrolled for all possible lengths of k' in L1 cache blocks, taking care to avoid overflowing the L1 instruction cache.
- *Re-buffering*: Since B' (Figure 11.5) is large (336×5) compared to A' (1×336), we deliberately buffer B' into L1 cache. While buffering B' we re-order its elements to enforce optimal memory access patterns. This has the additional benefit of minimising translation look-aside buffer (TLB) misses [Whaley et al., 2000]. The TLB is a cache of recently accessed virtual memory addresses.
- *Pre-fetching*: Values from A' are not deliberately pre-buffered into L1 cache. We make use of SSE pre-fetch assembler instructions to ensure A' values will be in L1 cache when needed.
- *Instruction re-ordering*: Despite the ability of modern CPUs to automatically re-order instructions, we found performance improved if care was taken to order assembler instructions so that processor stalls were minimised.
- *L2 Blocking*: Efficient L2 cache blocking ensures that peak rates can be maintained as long as A, B and C fit into main memory.

11.3.4 Emmerald Experimental Protocol

The performance of Emmerald was measured by timing matrix multiply calls with size $m = n = k$ up to 700. The following steps were taken to ensure a conservative performance estimate:

- wall clock time on an unloaded machine was used rather than CPU time;
- the stride of the matrices — the separation in memory between each row of matrix data — was fixed to 700 rather than the optimal value (the length of the row);
- caches were flushed between calls to `sgemm()`.

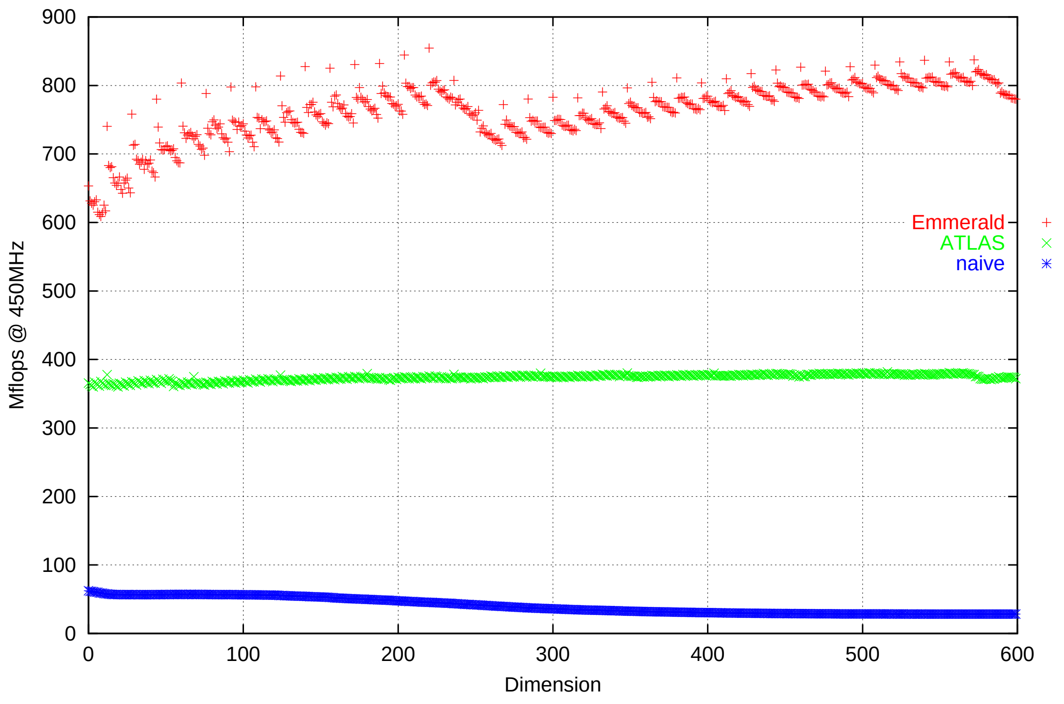


Figure 11.6: Performance of Emmerald on a PIII running at 450 MHz compared to `ATLAS` `sgemm` and a naive 3-loop matrix multiply. The version of `ATLAS` used in this figure did not make use of SIMD instructions.

11.3.5 Results

Timings were performed on a Pentium III @ 450 MHz running Linux (kernel 2.2.14). Figure 11.6 shows Emmerald's performance compared to `ATLAS` and a naive three-loop matrix multiply. The average MFlops/s rate of Emmerald after size 100 was 1.69 times the clock rate of the processor and 2.09 times faster than `ATLAS`. A peak rate of 890 MFlops/s was achieved when $m = n = k = stride = 320$. This represents 1.98 times the clock rate. On a PIII @ 550 MHz (the processors in Bunyip) we achieved a peak of 1090 MFlops/s. The largest tested size was $m = n = k = stride = 3696$ which ran at 940 MFlops/s @ 550 MHz. See the Emmerald papers Aberdeen and Baxter [2001], and Aberdeen and Baxter [2000], for more details.

11.4 Training Neural Networks using SGEMM

In this section we describe how to implement the error back propagation training algorithm efficiently using Emmerald. We assume a basic familiarity with the concepts of artificial neural network error back propagation. Detailed derivations of the procedure for single training vectors can be found in texts such as Haykin [1999]. This section simply extends those equations to multiple training vectors for one feed-forward and feed-back pass.

11.4.1 Matrix-Matrix Feed Forward

We assume a single hidden layer ANN mapping input vectors $y_t \in \mathbb{R}^{n_y}$ to output vectors $o_t \in \mathbb{R}^{n_o}$. Element i of the t'th input vector y_t is denoted y_{ti}. Similarly, element k of the t'th output o_t is labelled o_{tk}. The inputs are patterns or features for classification. In the POMDP setting they are observation vectors. The elements w_{ij} of the $n_y \times n_h$ matrix W is the weight from input feature y_{ti} to hidden unit j. Similarly, element v_{jk} of the $n_h \times n_o$ matrix V is the weight from unit j to output o_{tk}. The hidden layer is squashed by $\sigma : \mathbb{R} \to \mathbb{R}$ (we use tanh) which we define as an element-wise operator when applied to matrices.

Suppose we have n_p training patterns. The memory hierarchy optimisations described in the previous section mean it is more efficient to feed forward all training patterns simultaneously, doing 2 matrix-matrix multiplications, than to feed forward one training pattern at a time, doing $2n_p$ vector-matrix multiplications. So we define an $n_p \times n_y$ input matrix Y as having a row for each pattern

$$Y = \begin{bmatrix} y_{11} & \cdots & y_{1n_y} \\ \vdots & \ddots & \vdots \\ x_{n_p 1} & \cdots & x_{n_p n_y} \end{bmatrix}.$$

Similarly, for the output matrix we have

$$O = \begin{bmatrix} o_{11} & \cdots & o_{1n_o} \\ \vdots & \ddots & \vdots \\ o_{n_p 1} & \cdots & o_{n_p n_o} \end{bmatrix}.$$

With these definitions the matrix-matrix form of the feed forward computation is

$$O = \sigma(YW)V. \tag{11.1}$$

11.4.2 Error Back Propagation

For pattern classification we are given a target label for each training input $y_t \in \{y_1, \ldots, y_{n_p}\}$ of the form $l_t \in [-1, 1]^{n_o}$. For regression applications the target vector is

$l_t \in \mathbb{R}^{n_o}$. In matrix form

$$L = \begin{bmatrix} l_{11} & \dots & l_{1n_o} \\ \vdots & \ddots & \vdots \\ l_{n_p1} & \dots & l_{n_pn_o} \end{bmatrix}.$$

The goal is to find sets of parameters W and V minimising a cost function. Recall that we trained the LVCSR phoneme probability estimator by maximising the log probability (see Section 10.3.1.1). For the Japanese OCR problem we use the classical *mean squared error*:

$$\begin{aligned} e(W,V,Y,L) &= \frac{1}{2}\sum_{t=1}^{n_p}\sum_{c=1}^{n_o}(o_{tc} - l_{tc})^2 \\ &= \frac{1}{2}\sum_{t=1}^{n_p}(\|o_t - l_t\|_2^2) \end{aligned}$$

Let be $H := \sigma(YW)$ be the matrix of hidden layer activations with elements h_{tj}. We also define the error at the output as the $n_p \times n_o$ matrix Δ_O, and the so-called back-propagated error at the hidden layer as the $n_p \times n_h$ matrix Δ_H

$$\begin{aligned} \Delta_O &= O - L, \\ \Delta_H &= (Y - H * H) * (\Delta_O V), \end{aligned} \tag{11.2}$$

where "$*$" denotes element-wise matrix multiplication. Following these conventions, the standard formulae for the gradients in matrix form are

$$\begin{aligned} \nabla^W e(W,V,Y,L) &= Y'\Delta_H \\ &= \begin{bmatrix} \frac{\partial e}{\partial w_{11}} & \dots & \frac{\partial e}{\partial w_{1n_h}} \\ \vdots & \ddots & \vdots \\ \frac{\partial e}{\partial w_{n_y1}} & \dots & \frac{\partial e}{\partial w_{n_yn_h}} \end{bmatrix} \\ \nabla^V e(W,V,Y,L) &= H'\Delta_O \\ &= \begin{bmatrix} \frac{\partial e}{\partial v_{11}} & \dots & \frac{\partial e}{\partial v_{1n_o}} \\ \vdots & \ddots & \vdots \\ \frac{\partial e}{\partial v_{n_h1}} & \dots & \frac{\partial e}{\partial v_{n_hn_o}} \end{bmatrix}. \end{aligned} \tag{11.3}$$

An equivalent form of matrix-matrix error back propagation was derived by Bilmes et al. [1997].

Thus, computing the gradient of the error for an ANN can be reduced to a series of ordinary matrix multiplications and element-wise matrix operations. For large networks and large numbers of training patterns, the bottleneck is the ordinary matrix multiplications. We implement these using Emmerald, dramatically speeding up the feed-forward and feed-backward stages. In all our experiments we found 32 bits of floating-point precision were enough for training. For neural networks with $\approx$ 10,000 parameters, as few as 16 bits are sufficient [Asanović and Morgan, 1991].

Armed with the gradient ∇e, the same parameter optimisation procedure used throughout this thesis is performed: a combination of the Polak-Ribiére conjugate-gradient method with the GSEARCH line search to optimise the weights, described in detail in Appendix B.1. We also found that quadratic penalties were important when training ULSNNs since the parameterisation is extremely rich and the potential for stopping in a poor local maximum is large.

11.4.3 Training Set Parallelism

Since the error e and gradient ∇e are *additive* over the training examples, the simplest way to parallelise the training of a neural network is to partition the training data into disjoint subsets and have each processor compute the error and gradient for its subset. This works particularly well if there are a large number of training patterns so that each processor can work with near-optimal matrix sizes.

The communication required is the transmission of the neural network parameters to each slave processor, the transmission of the error and gradient information back from each slave to a master node, and the broadcast of the next search direction θ^* back to the slaves so they have a consistent notion of the search direction.

For small ANNs with 100's to 10,000's of parameters the communication costs are negligible. However, with millions of parameters optimisation of communication can result in significant speed-ups. The most expensive communication step is aggregating the gradient vectors from each slave to the master. This is known as a *reduce* operation and involves 194 CPUs each sending 6.6 Mbytes of data to the master process. We spent some time coding an optimised reduce operation for the Bunyip cluster. The ideas can be translated to other clusters that use a *flat network topology* [Hauser et al., 2000], where all nodes are connected to all others through at most one switch. The details of the communication costs and our optimised reduce operation are in Appendix G.

If we are training a POMDP agent then training patterns are generated by simulating the environment. In this case each processor runs its own simulation for a fixed amount of time before sending back its gradient estimate to the server. Unfortunately, the feed-forward operation cannot be blocked into multiple patterns because the output of the network is needed to generate the next input.

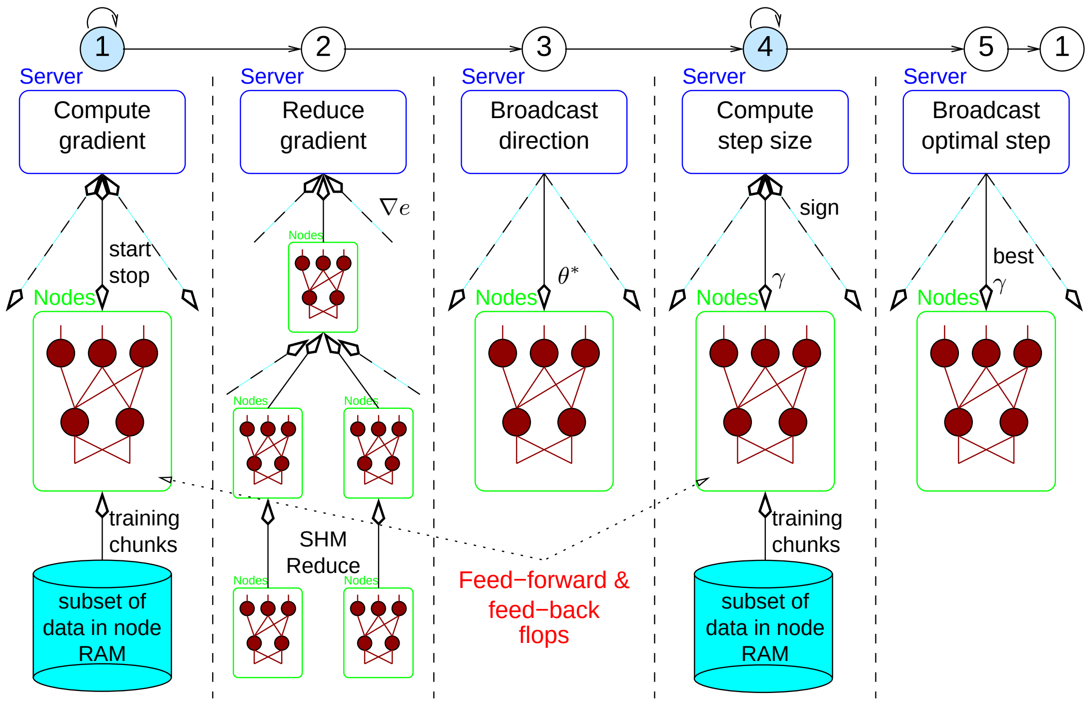


Figure 11.7: Steps followed for each epoch of ULSNN training. Arrows represent communication directions, which are labelled with the messages being communicated.

11.5 Using Bunyip to Train ULSNNs

With reference to Figure 11.7, distributed training of ULSNNs on Bunyip progresses over 5 stages per epoch.

1. Section 11.4 describes stage 1, with each process computing the gradient induced by a subset of the training data.

2. Appendix G.2 describes how the gradient is reduced from each slave process to the master process.

3. This stage is the relatively cheap broadcast of the search direction to each process.

4. During this stage we broadcast scalar trial steps sizes to the slaves, and reduce the scalar results back to the master. This stage implements the GSEARCH algorithm described in Appendix B.1.

5. The final step before repeating the process is the broadcast of the final step size, γ, determined by GSEARCH. Each processor then updates its parameters γ.

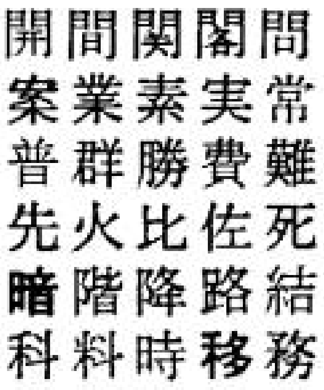


Figure 11.8: Example Japanese characters used to train the ULSNN. The characters in each row are different but look similar, illustrating the difficulty of Japanese OCR.

When calculating the price/performance ratio we only count the flops performed during feed-forward and feed-back computations, during phases 2 and 4. We do not count the flops performed during the reduce or any other phases.

11.6 Japanese Optical Character Recognition

This section describes an application of our SIMD-optimised distributed ANN training. We train an ULSNN as a classifier for machine-printed Japanese characters.

Japanese optical character recognition (Japanese OCR) is the process of automatically recognizing machine-printed Japanese documents. The most difficult aspect of Japanese OCR is correctly classifying individual characters, since there are approximately 4000 characters in common usage.

11.6.1 Experimental Protocol

The base training data for our neural network consisted of 168,000 scanned, segmented, hand-truthed images of Japanese characters purchased from the CEDAR group at the University of Buffalo [Srihari et al., 1996]. The characters were scanned from a variety of sources, including books, faxes, newspapers and magazines. Figure 11.8 gives an idea of the varying quality of the character images.

Each character in the CEDAR database is represented as a binary image of varying resolution. We down-sampled all the images to a 20×20 grey-scale format. The neural network had 400 input nodes, one for each pixel. The database contained examples of 3203 distinct characters, hence the neural-network had 3203 output nodes. The hidden layer was chosen to have 480 nodes. In total, the network had 1.73 million parameters.

Using 168,000 training examples is not sufficient to avoid over-fitting in a network containing 1.73 million adjustable parameters. We generated synthetic data from the original characters by applying random transformations including line thickening and thinning, shifting, blurring, and noise addition. The total number of training examples including the artificial ones was 9,264,000, approximately 5.4 per adjustable parameter. These were distributed uniformly to the processors in Bunyip. A further 6,320 examples of the CEDAR data set were used for testing purposes.

The weights were initialised randomly between [-0.1, 0.1]. The quadratic penalty was set to $\wp = 0.1$.

With reference to equations (11.1) – (11.3), the total number of floating point operations required to compute the error e in a neural network is $2n_p(n_y+n_o)n_h$, which equals 32 Tera floating-point operations (TFlops) for the Japanese OCR experiment. A gradient calculation uses $n_p(4n_yn_h + 6n_hn_o)$ flops, or 92 TFlops for the Japanese OCR experiment.

To assist with load balancing, each slave processor stepped through its training patterns 320 at a time. Between each step the master node was polled to determine whether more steps were required. Once 80% of the total training data had been consumed, the master instructed all slaves to halt computation and return their results (either the error or the gradient). In this way the idle time spent waiting for other slaves to finish was reduced to at most the length of time needed by a single processor to process 320 patterns. Accounting for the fact that only 80% of the data is processed, an error calculation required 26 TFlops and a gradient calculation requires 74 TFlops, or 135 GFlops and 383 GFlops per processor respectively.

11.6.2 Results

This section describes the classification accuracy achieved, then concentrates on the performance scalability over processors, before finishing with the peak performance results that justify our claim of a price/performance ratio of 92.4 per MFlop/s.

11.6.2.1 Classification Accuracy

The network's best classification error on the held-out 6,320 examples was 33%, indicating substantial progress on a difficult problem (an untrained classifier has an error of $1 - 1/3200 = 99.97\%$). We observed an error rate of 5% on the 40% of the data that contained the most common characters.

A very large amount of data was required to avoid over-fitting. Table 11.1 compares the generalisation accuracy against the total number of training examples used (including transformations of the original 168,000 patterns).

Table 11.1: Generalisation error for Japanese OCR decreases as the total number of patterns increases.

Patterns	*% error*
343800	51
611200	46
1833600	33

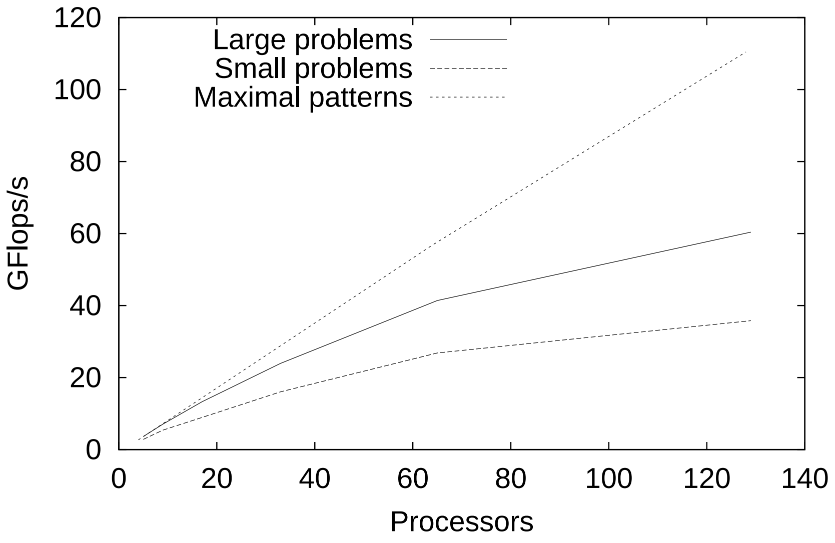


Figure 11.9: Performance scaling with the number of processors. The 3 curves represent a small network, our large Japanese OCR network with a fixed number of patterns, and the Japanese OCR network when n_p scales with the number of processors.

11.6.2.2 Communication Performance

The primary motivation for large training sets is to improve generalisation performance. In Appendix G.1 we demonstrate that communication overhead increases as the number of patterns decreases, so the second motivation for large training sets is to reduce such overhead. Figure 11.9 demonstrates how the performance scales with the number of processors used. The bottom curve is the performance versus processors curve for a small network of 400 input nodes, 80 hidden layer nodes, 200 output nodes and a total of 40,960 training patterns. The middle curve is our Japanese OCR ULSNN with 163,480 patterns in total. The top curve is the Japanese OCR network again, however, this test used 32,000 patterns *per processor*, minimising the frequency of reduce operations.

All 3 curves exhibit linear performance scaling for small numbers of processors, but for many processors the larger problem scales better despite the increased number of network parameters. This is due to the communication overhead in the small network increasing dramatically as each processor has less data to process before needing to initiate a reduce. The effect would be clearer for a large network (causing long gradient vectors to be reduced) with few training patterns, however this scenario is not usually encountered due to over-fitting. Finally, we observe that with a large enough data set to fill the memory of every node, we achieve near linear scaling.

11.6.2.3 Price/Performance Ratio

All performance values quoted in this section represent the total flops that contribute to feed forward and gradient calculations, divided by the wall clock time. Implementation specific flops, such as the reduce operations, were not included.

Bunyip was dedicated to running the Japanese OCR problem for 4 hours with 9,360,000 patterns distributed across 196 processors. Bunyip actually consists of 194 processors, however, we co-opted one of the hot-spare nodes (included in the quoted price) to make up the other two processors.

Over this 4 hour period a total of 2.35 PFlops were performed with an average performance of 163.3 GFlops/s. This performance is sustainable indefinitely provided no other processes use the machine. To calculate the price/performance ratio we used the total cost derived in Section 11.2.2 of USD \$150,913, yielding a ratio of 92.4 per MFlop/s.[3]

For comparison purposes, training using double precision and the ATLAS DGEMM [Whaley and Dongarra, 1997] produced a sustained performance of 70 MFlops/s or \$2.16 per MFlops/s (double precision).

11.7 Summary

Key Points

I We have shown how an off-the-shelf Linux Pentium III cluster can efficiently train ultra large scale neural networks. This is a particularly cost-effective solution with a price/performance ratio of 92.4 per MFlop/s.

II Such clusters are a good choice for Monte-Carlo POMDP methods since each processor can run an independent simulation. Communication is only needed to occasionally reduce gradient estimates and update parameters.

[3]Based on the exchange rate of AUD \$1 = USD .5965 on the day of the final and largest payment.

Fast matrix-matrix multiplication significantly speeds up training of large neural networks such as those used in our LVCSR experiment.

Subsequent Work

When Emmerald was written it was, within our knowledge, the fastest SGEMM for the Pentium III processor. Some of the ideas from Emmerald were incorporated into earlier versions of the self-optimising `ATLAS` BLAS library [Whaley et al., 2000]. `ATLAS` has now surpassed the performance of Emmerald, also incorporating double-precision GEMM.

The Gordon Bell prize for price/performance ratio has followed a trend of almost quartering the cost per MFlop/s each year. The 2001 award was won by Kim et al. [2001], with a cluster that achieved 24.6 per MFlops/s.

Chapter 12

Conclusion

When we write programs that "learn," it turns out that we do and they don't.

—Alan Perlis

We have demonstrated that policy-gradient methods can be used to train memory-enabled agents so that they perform well in complex domains. We investigated domains such as simulated robot navigation and speech recognition.

12.1 Key Contributions

The key contributions of this thesis are:

- I The GAMP, IState-GPOMDP, IOHMM-GPOMDP, and Exp-GPOMDP, policy-gradient algorithms for estimating long-term average reward gradients. They allow agents with FSC memory to be trained in infinite-horizon settings.
- II Higher-order eligibility trace filters and other methods for reducing the variance of gradient estimates.
- III The application of POMDP methods to the complex real-world domain of speech recognition.

The use of approximations such as local optimisation, finite memory, and iterative approximations, allowed us to train agents for difficult scenarios that could not be solved with exact methods. Previous FSC policy-gradient algorithms also failed to scale to non-trivial scenarios.

12.2 Long-Term Future Research

Each chapter has mentioned specific future research projects. To end, we present a vision of how POMDP agents might be trained for real applications.

For most purposes real-world problems have infinite-state spaces, and often infinite-observation spaces, making methods that track *exact* belief states impossible. Factoring state into state-variables is an active and interesting area of research [Poupart and

Boutilier, 2001]. How to perform such factorisation without a model or for infinite spaces is not clear.

Methods that estimate values of belief states are difficult to scale to real scenarios for two reasons: (1) the complexity of belief-state tracking when the state space is infinite, (2) the extra effort needed to estimate values compared to learning the policy directly. On the other hand, policy-gradient methods suffer from slow convergence and convergence to only a *local* maximum. However, the PSPACE-hard complexity of finding the globally optimal policy is the best argument for the use of local optimisation approaches, and current and future research is improving the convergence rates for policy-gradient methods.

Assuming policy-gradient methods are the best approach for large problems, how should we implement memory? This thesis has advocated finite state controllers because of their ability to recall events indefinitely far into the past. However, learning FSCs becomes more difficult as we increase the number of I-states. The best approach may be to combine FSC methods with finite history methods, which is easily accomplished using policy-gradient algorithms. This is intuitively appealing since humans make the distinction between detailed short-term memory, and long-term memory, which only stores important information.

The biggest problem with policy-gradient approaches is the amount of experience and time needed to converge to a good policy. Also, it is often impractical to allow an agent to execute a random policy, as they must without any prior knowledge of the domain. For example, consider learning to drive a car. We will not allow an agent to control a real car until we are confident it can already drive without crashing. One solution is to simulate the world on a computer for early training. This allows the use of variance reduction methods such as fixed random sequences.

If we can construct a simulator, even an approximate one, we can construct a factored-state model of the POMDP. Research on the impact of approximating MDPs and POMDPs suggests that reasonable real-world behaviour is assured if the approximation is done appropriately [Poupart and Boutilier, 2001, Guestrin et al., 2001b]. If algorithms such as GAMP can be extended to work with factored POMDP models then we will have a powerful tool for generating complex-domain agents quickly. Once GAMP has been used to train an agent capable of driving reasonably well, the agent can be fine-tuned using real-world experience. During the latter phase we still wish to minimise the amount of experience needed. Thus, we implement methods such as importance-sampling for off-policy training [Shelton, 2001b] and additive control variates [Greensmith et al., 2002].

In summary , training for complex-domains should:

1. construct an agent with parameterised stochastic policy, parameterised stochastic

FSC, and a finite history window;

2. add domain knowledge to the agent by coding rules that shape the output distributions of I-state transitions and actions;

3. construct an approximate factored-state model of the scenario;

4. use factored-state GAMP to generate gradients for training the agent to act in the simulated domain;

5. use IState-GPOMDP or Exp-GPOMDP, along with variance reduction methods, to fine tune the agent by interacting with the real world.

The world is a POMDP. By studying methods in the POMDP framework we gain confidence in our ability to train machines to perform in any domain.

Glossary of Symbols

These are the symbols used consistently through the thesis. Some symbols may sometimes be subscripted with t, denoting the value of the symbol at time t.

$\mathcal{S}$	World state space of finite size $\|\mathcal{S}\|$	11
$\mathcal{U}$	Action space of finite size $\|\mathcal{U}\|$	11
$\mathcal{Y}$	Observation space of finite size $\|\mathcal{Y}\|$	11
i	Current world state in $\mathcal{S}$	11
$r(i)$	Reward for being in state i	11
r_t	Scalar reward at time t	11
$\nu(y\|i)$	Probability of observation y given state i	12
$q(j\|i,u)$	Probability of transition $i \to j$ under action u	12
j	Next world state	12
u	Action in $\mathcal{U}$	12
y	Observation in $\mathcal{Y}$	12
$\mu(u\|\theta,h,y)$	Policy: conditional probability of action $u \in \mathcal{U}$	14
n_ϕ	Number of parameters controlling internal state	14
n_θ	The number of policy parameters	14
$\omega(h\|\phi,g,y)$	Finite state controller: conditional probability of I-state $h \in \mathcal{G}$	14
$\mathcal{G}$	Set of internal states of size $\|\mathcal{G}\|$	14
ϕ	Internal state parameters in $\mathbb{R}^{n_\phi}$	14
θ	Policy parameters in $\mathbb{R}^{n_\theta}$	14
g	Current internal state in $\mathcal{G}$	14
h	Next internal state in $\mathcal{G}$	14
$J_\beta(\phi,\theta,i,g)$	Discounted sum of rewards	15
β	Discount factor in $[0,1)$	15
$\eta(\phi,\theta,i,g)$	Long-term average reward	15
$\chi_m(k)$	Indicator function	17
ϵ	A small value used as a convergence threshold	17

$Q(i,u)$	discounted value of action $u \in \mathcal{U}$ in state $i \in \mathcal{S}$	18
γ	Step size for parameter updates	18
$b(i\|\bar{y}_t)$	Belief state: conditional probability of world state $i \in \mathcal{S}$	19
$\mathcal{B}$	Set of reachable belief states	19
$\bar{y}$	Vector of all past observations up to the current time	19
$\bar{J}_\beta(b)$	Discounted value of belief state $b \in \mathcal{B}$	20
$\mathcal{L}$	Set of $\|\mathcal{S}\|$ dimensional hyperplanes	20
v	Number of state variables	22
$u^*(i)$	Action chosen deterministically for state $i \in \mathcal{S}$ or $b \in \mathcal{B}$	23
$P(\phi,\theta)$	Global-state Markov chain stochastic matrix	47
$p(j,h\|\phi,\theta,i,g)$	Entries of P	47
c	Index into the θ parameters $c \in \{1,\dots,n_\theta\}$	48
l	Index into the ϕ parameters $l \in \{1,\dots,n_\phi\}$	48
R	Maximum absolute reward	49
$\pi(\phi,\theta)$	Stationary distribution vector with $\|\mathcal{S}\| \times \|\mathcal{G}\|$ elements	49
$\pi(i,g\|\phi,\theta)$	Stationary distribution vector elements	49
o	Artificial neural network output vector	52
e	Column vector of $\|\mathcal{S}\| \times \|\mathcal{G}\|$ ones	56
x_N	Order N Richardson approximation to $[I - P + e\pi']^{-1} r$	59
τ	Mixing time of the POMDP	60
Δ_T	Gradient estimate after T steps of a Monte-Carlo algorithm	71
z	Eligibility trace vector of length $n_\phi + n_\theta$	72
$\xi(m\|\phi_\xi,g)$	Emission probability of symbol m given I-state $g \in \mathcal{G}$	79
$\alpha_t(g\|\phi,\bar{y}_{t-1})$	Conditional probability of I-state $g \in \mathcal{G}$ for Exp-GPOMDP	86
k	Number of transitions out of any I-state $g \in \mathcal{G}$ for given $y \in \mathcal{Y}$	95
τ	Delay between actions and rewards for IIR trace filter	117
a_n, b_n	IIR eligibility-trace filter co-efficients	119
$\mathcal{M}$	Set of models (HMMs or phonemes) of size $\|\mathcal{M}\|$ models	134
m	A HMM or phoneme from $\mathcal{M}$	134
$\hat{\alpha}_t(h\|\phi,\bar{y}_{t-1})$	Conditional probability of I-state h using Viterbi procedure	147
θ^*	Gradient ascent search direction	209
$\wp$	Scalar quadratic penalty value	213

Appendix A

Proofs

A.1 GAMP Proofs

This section contains the proofs for the Theorems stated in Chapter 4.

A.1.1 Proof of Theorem 2

For brevity, the following proof is for the θ derivatives in the case of a memory-less agent — the FSC memory case is similar.

Proof. From (4.8) we see that

$$\begin{aligned}
\|\widehat{\nabla_N \eta} - \nabla\eta\|_\infty &= \max_{\theta_c} \left| \pi' \frac{\partial P}{\partial \theta_c} \sum_{n=N}^{\infty} P^n r \right| \\
&\leq \max_{\theta_c} \|\pi'\|_1 \left\| \sum_{n=N}^{\infty} \frac{\partial P}{\partial \theta_c} P^n r \right\|_\infty, \quad \text{Hölder's inequality} \\
&\leq \max_{\theta_c} \|\pi'\|_1 \sum_{n=N}^{\infty} \left\| \frac{\partial P}{\partial \theta_c} P^n r \right\|_\infty, \quad \text{Triangle inequality,} \\
&= \max_{\theta_c} \sum_{n=N}^{\infty} \left\| \frac{\partial P}{\partial \theta_c} P^n r \right\|_\infty, \quad \|\pi'\|_1 = 1. \qquad \text{(A.1)}
\end{aligned}$$

Now we concentrate on the n'th term of the summation. In the following expression the vector ξ_n takes the place of $P^n r$ and ξ_{nj} is the j'th element of ξ_n. We now assume all norms are L_∞.

$$\begin{aligned}
\left\| \frac{\partial P}{\partial \theta_c} P^n r \right\| &= \left\| \frac{\partial P}{\partial \theta_c} \xi_n \right\| \\
&= \left\| \sum_{j,y,u} \nu(y|i) \frac{\partial \mu(u|\theta, y)}{\partial \theta_c} q(j|i,u) \xi_{nj} \right\|. \qquad \text{(A.2)}
\end{aligned}$$

Now we show the computation for each element of ξ_n, defining it in terms of η and the variation of ξ_n away from η. Let $p_{ij}^{(n)}$ be the row i column j element from P^n.

Furthermore, let $\Delta_n(i,j) := p_{ij}^{(n)} - \pi_j$, so that

$$\begin{aligned}
\xi_{ni} &= \sum_j p_{ij}^{(n)} r_j \\
&= \sum_j (\pi_j + (p_{ij}^{(n)} - \pi_j)) r_j \\
&= \sum_j (\pi_j + \Delta_n(i,j)) r_j \\
&= \eta + \sum_j \Delta_n(i,j) r_j .
\end{aligned}$$

Substituting back into Equation (A.2) we obtain

$$\begin{aligned}
&\left\| \sum_{j,y,u} \nu(y|i) \frac{\partial \mu(u|\theta,y)}{\theta_c} q(j|i,u) \xi_{nj} \right\| \\
&= \left\| \sum_{j,y,u} \nu(y|i) \frac{\partial \mu(u|\theta,y)}{\partial \theta_c} q(j|i,u) \left(\eta + \sum_m \Delta_n(j,m) r_m \right) \right\| \\
&= \left\| \sum_{j,y,u} \nu(y|i) \frac{\partial \mu(u|\theta,y)}{\partial \theta_c} q(j|i,u) \eta + \right. \\
&\qquad \left. \sum_{j,y,u} \nu(y|i) \frac{\partial \mu(u|\theta,y)}{\partial \theta_c} q(j|i,u) \sum_m \Delta_n(j,m) r_m \right\| .
\end{aligned}$$

The first term in the last line represents the gradient contribution assuming P^n has converged to π, which we now show is zero. The second term represents the error due to the fact that P^n is not exactly π

$$\begin{aligned}
&= \left\| \eta \sum_y \nu(y|i) \sum_u \frac{\partial \mu(u|\theta,y)}{\partial \theta_c} \mu(u|\theta,y) \underbrace{\sum_j q(j|i,u)}_{=1} + \right. \\
&\qquad \left. \sum_{j,y,u} \nu(y|i) \frac{\partial \mu(u|\theta,y)}{\partial \theta_c} q(j|i,u) \sum_m \Delta_n(j,m) r_m \right\| \\
&= \left\| \eta \sum_y \nu(y|i) \underbrace{\frac{\partial}{\partial \theta_c} \sum_u \mu(u|\theta,y)}_{=0} + \right. \\
&\qquad \left. \sum_{j,y,u} \nu(y|i) \frac{\partial \mu(u|\theta,y)}{\partial \theta_c} q(j|i,u) \sum_m \Delta_n(j,m) r_m \right\| ,
\end{aligned}$$

so we are left with only the second term in the norm. We compute the norm by taking the maximum over i and taking the absolute values of $\frac{\partial\mu(u|\theta,y)}{\partial\theta_c}$ and $\sum_m \Delta_n(j,m)$, which are the only possibly negative quantities

$$\leq \max_i \sum_{y,u} \nu(y|i) \left|\frac{\partial\mu(u|\theta,y)}{\partial\theta_c}\right| \sum_j q(j|i,u) \left|\sum_m \Delta_n(j,m) r_m\right|.$$

Recall from Equation (A.1) that we need to take the maximum over all parameters θ_c. We now perform this step by recalling that the maximum derivative with respect to any parameter is bounded by U (see Assumption 3, Section 3.2)

$$\begin{aligned}\max_{\theta_c} \sum_{n=N}^{\infty} \max_i \sum_{y,u} \nu(y|i) \left|\frac{\partial\mu(u|\theta,y)}{\partial\theta_c}\right| \sum_j q(j|i,u) \left|\sum_m \Delta_n(j,m) r_m\right| \\ \leq U \sum_{n=N}^{\infty} \max_i \sum_{y,u} \nu(y|i) \sum_j q(j|i,u) \left|\sum_m \Delta_n(j,m) r_m\right|.\end{aligned}$$

Furthermore, the reward is bounded by R (see Assumption 2, Section 3)

$$\leq UR \sum_{n=N}^{\infty} \max_i \sum_{y,u} \nu(y|i) \sum_j q(j|i,u) \left|\sum_m \Delta_n(j,m)\right|. \tag{A.3}$$

Now observe that

$$\begin{aligned}\left|\sum_m \Delta_n(j,m)\right| &\leq \sum_m |\Delta_n(j,m)|, \quad \text{Triangle inequality} \\ &= \sum_m |p_{jm}^{(n)} - \pi_m| \\ &= d(P_j^n, \pi),\end{aligned}$$

that is, the total variation distance from Definition 3. Returning to Equation (A.3) and substituting the inequality above we obtain

$$\leq UR \sum_{n=N}^{\infty} \max_i \sum_{y,u} \nu(y|i) \sum_j q(j|i,u) d(P_j^n, \pi),$$

which is further simplified by letting $\bar{d}(n) = \max_j d(P_j^n, \pi)$, giving

$$\begin{aligned}
&\leq UR \sum_{n=N}^{\infty} \max_i \sum_{y,u} \nu(y|i) \sum_j q(j|i,u) \bar{d}(n) \\
&= UR \sum_{n=N}^{\infty} \max_i \bar{d}(n) \sum_{y,u} \nu(y|i) \underbrace{\sum_j q(j|i,u)}_{=1} .
\end{aligned}$$

Next we eliminate all references to the state i, making the $\max_i$ operator redundant

$$\begin{aligned}
&= UR \sum_{n=N}^{\infty} \max_i \bar{d}(n) \underbrace{\sum_{y,u} \nu(y|i)}_{=|\mathcal{U}|} \\
&= UR|\mathcal{U}| \sum_{n=N}^{\infty} \bar{d}(n).
\end{aligned}$$

We have shown that the resultant vector indexed by i is uniform and the norm is obtained by taking any element $i \in \mathcal{S}$. Now we make use of Definition 4 (see Section 4.3), bounding $\bar{d}(n)$ in terms of the mixing time τ of the stochastic matrix P

$$\begin{aligned}
&\leq UR|\mathcal{U}| \sum_{n=N}^{\infty} \exp(-\left\lfloor \frac{n}{\tau} \right\rfloor) \\
&= UR|\mathcal{U}| \left[\left(\sum_{l=\left\lfloor \frac{N}{\tau} \right\rfloor + 1}^{\infty} \sum_{k=0}^{\tau-1} \exp(-l) \right) + \sum_{k=N \mod \tau}^{\tau-1} \exp(-\left\lfloor \frac{N}{\tau} \right\rfloor) \right] \qquad \text{(A.4)} \\
&\leq UR|\mathcal{U}| \sum_{l=\left\lfloor \frac{N}{\tau} \right\rfloor}^{\infty} \sum_{k=0}^{\tau-1} \exp(-l). \qquad \text{(A.5)}
\end{aligned}$$

The last two lines re-write the bound so that the floor operator does not appear in the summation. We arrive at Equation (A.4) by re-writing the first line as a sum of sums where all terms in the second summation have constant $l = \lfloor n/\tau \rfloor$. The second term of Equation (A.4) is the last part of the summation for which there are not exactly τ terms. It is eliminated by noting that $k = N \mod \tau \leq \tau$ and combining it with the first sum by subtracting one from the initial summation index l. We now observe that we are performing the first sum τ times, and apply a standard series convergence result

$$\begin{aligned}
&= UR|\mathcal{U}|\tau \sum_{l=\lfloor \frac{N}{\tau} \rfloor}^{\infty} \exp(-l) \\
&= UR|\mathcal{U}|\tau \sum_{l=\lfloor \frac{N}{\tau} \rfloor}^{\infty} \left(\frac{1}{\exp(1)} \right)^{l} \\
&= UR|\mathcal{U}|\tau \frac{\exp(-\lfloor \frac{N}{\tau} \rfloor)}{1-\exp(-1)}.
\end{aligned}$$

□

A.1.2 Proof of Theorem 3

We begin with the following basic lemma.

Lemma 1. *Let P be a stochastic matrix and r a column vector.*

$$\|P^{N+1}r\|_\infty \leq \|P^N r\|_\infty.$$

Proof. Let $p_{ij}^{(N)}$ be the row i, column j entry of P^N.

$$\begin{aligned}
\|P^{N+1}r\|_\infty &= \|PP^N r\|_\infty \\
&= \max_i \left| \sum_c \sum_j p_{ic} p_{cj}^{(N)} r_j \right| \\
&= \max_i \sum_c p_{ic} \left| \sum_j p_{cj}^{(N)} r_j \right| \\
&\leq \max_i \sum_c p_{ic} \|P^N r\|_\infty \\
&= \|P^N r\|_\infty \max_i \underbrace{\sum_c p_{ic}}_{=1} \\
&= \|P^N r\|_\infty.
\end{aligned}$$

□

Proof of Theorem 3. This follows immediately from Lemma 1 and the observation that successive evaluations of Equation (4.11) give us for all N

$$\begin{aligned}\|x_{N+1} - x_N\|_\infty &= \|P^{N+1}r\| \\ \|x_N - x_{N-1}\|_\infty &= \|P^N r\|.\end{aligned}$$

□

A.2 IState-GPOMDP Proofs

This section contains proofs of the theorems presented in Chapter 5.

A.2.1 Proof of Theorem 5

The following proof is copied from the original GPOMDP paper [Baxter and Bartlett, 2001]. The proof follows through unchanged for the FSC version because we consider P to the global-state matrix that combines the I-states and world-states into one MDP. The first part of the proof has already been established in Section 4.1 so we review it quickly.

Proof. Recall from Equation (4.6) that

$$\nabla\eta = \pi'(\nabla P)\left[I - (P - e\pi')\right]^{-1} r.$$

A quick induction argument given by (4.5) shows that $[P - e\pi']^n = P^n - e\pi'$ which, given the assumptions of Section 3.2, converges to 0 as $n \to \infty$. So by a classical matrix theorem, $[I - (P - e\pi')]^{-1}$ exists and is equal to $\sum_{n=0}^{\infty}[P^n - e\pi']$. Observe that $(\nabla P)e\pi' = 0$ so Equation (4.6) can be rewritten as

$$\nabla\eta = \pi'\left[\sum_{n=0}^{\infty}(\nabla P)P^n\right] r. \tag{A.6}$$

Now let $\beta \in [0, 1)$ be a discount factor and consider the following modification to Equation (A.6)

$$\widehat{\nabla_\beta}\eta := \pi'\left[\sum_{n=0}^{\infty}(\nabla P)(\beta P)^n\right] r. \tag{A.7}$$

Since $\lim_{\beta\to 1}\nabla_\beta\eta = \nabla\eta$, and since $(\beta P)^n = \beta^n P^n \to \beta^n e\pi' \to 0$, we can take ∇P back out of the sum and write

$$\widehat{\nabla_\beta \eta} = \pi'(\nabla P)\left[\sum_{n=0}^{\infty}\beta^n P^n\right] r. \tag{A.8}$$

But $[\sum_{n=0}^{\infty}\beta^n P^n]\, r = J_\beta$, where $J_\beta = [J_\beta(1,1), \dots, J_\beta(|\mathcal{S}|, |\mathcal{G}|)]$ is the vector of expected discounted rewards from each world/I-state pair, as defined by Equation (2.2). This gives us the required expression

$$\nabla\eta = \lim_{\beta\to 1}\pi'(\nabla P)J_\beta.$$

□

A.2.2 Proof of Theorem 4

This proof is an easy generalisation of Baxter and Bartlett [2001, Thm. 4].

Proof. Let $\{S_t\}$ denote the random process corresponding to the world-state Markov chain generated by $P(\phi, \theta)$. Also, let $\{G_t\}$ denote the random process corresponding to the I-state Markov chain generated by $P(\phi, \theta)$ (see Equation (3.1)). By Assumption 1 (see Section 3.2), $\{S_t, G_t\}$ is asymptotically stationary, and we can write

$$\begin{aligned}
\pi'(\nabla P)J_\beta &= \sum_{i,j,g,h}\pi(i,g)\nabla p(j,h|\phi,\theta,i,g)J_\beta(j,h), \qquad \text{now apply (3.1)}\\
&= \sum_{i,j,y,u,g,h}\pi(i,g)q(j|i,u)\nu(y|i)\\
&\qquad \Big[\omega(h|\phi,g,y)\nabla^\theta\mu(u|\theta,h,y), \nabla^\phi\omega(h|\phi,g,y)\mu(u|\theta,h,y)\Big]\, J_\beta(j,h)\\
&= \sum_{i,j,y,u,g,h}\pi(i,g)q(j|i,u)\nu(y|i)\\
&\qquad \Bigg[\omega(h|\phi,g,y)\frac{\nabla^\theta\mu(u|\theta,h,y)}{\mu(u|\theta,h,y)}\mu(u|\theta,h,y)J_\beta(j,h),\\
&\qquad\quad \frac{\nabla^\phi\omega(h|\phi,g,y)}{\omega(h|\phi,g,y)}\omega(h|\phi,g,y)\mu(u|\theta,h,y)J_\beta(j,h)\Bigg].
\end{aligned}$$

There are two independent sets of gradient values for the ϕ and θ parameters. The remainder of the proof follows the gradient w.r.t. ϕ, however, the proof w.r.t. θ follows the same process. Dropping the ∇^θ components, we can rewrite the last expression as

$$\sum_{i,j,y,u,g,h}\mathbb{E}_{\phi,\theta}\chi_i(S_t)\chi_j(S_{t+1})\chi_g(G_t)\chi_h(G_{t+1})\chi_u(U_t)\chi_y(Y_t)\frac{\nabla\omega(h|\phi,g,y)}{\omega(h|\phi,g,y)}J(t+1), \tag{A.9}$$

where $\chi_i(X)$ denotes the indicator function (2.6), and the expectation is with respect to the joint distribution of $\{S_t, S_{t+1}, Y_t, G_t, G_{t+1}, U_t\}$ when S_t is stationary and parameterised by ϕ, and θ. Here $J(t+1)$ is the process governing the discounted infinite-horizon reward for a single sample trajectory

$$J(t+1) = \sum_{s=t+1}^{\infty} \beta^{s-t-1} r(S_s), \tag{A.10}$$

such that

$$J_\beta(\phi, \theta, j, h) = \mathbb{E}_{\phi,\theta}[J(t+1)|S_{t+1} = j, G_{t+1} = h]$$

follows from the boundedness of the rewards and Lebesgue's dominated convergence theorem. This matches our definition of J_β from Equation (2.2).

We have assumed $\{S_t, G_t\}$ is stationary and ergodic, thus the joint process $Z_t = \{S_t, S_{t+1}, Y_t, G_t, G_{t+1}, U_t\}$ is stationary and ergodic (because Y_t is i.i.d. given S_t, G_{t+1} is i.i.d. given G_t and Y_t, U_t is i.i.d. given G_{t+1} and Y_t, and S_{t+1} is i.i.d. given S_t and U_t). Since Z_t is obtained by taking a fixed function of $\{S_t, S_{t+1}, Y_t, G_t, G_{t+1}, U_t\}$, it is also stationary and ergodic (see [Breiman, 1966, Proposition 6.31]). As $\left\| \frac{\nabla\omega(h|\phi,g,y)}{\omega(h|\phi,g,y)} \right\|$ is bounded by Assumption 4, from the Ergodic Theorem we obtain with probability 1

$$\begin{aligned}
&\pi'(\nabla^\phi P)J_\beta = \\
&\sum_{i,j,y,u,g,h} \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \chi_{ijuygh}(S_t, S_{t+1}, U_t, Y_t, G_t, G_{t+1}) \frac{\nabla\omega(h|\phi, g, y)}{\omega(h|\phi, g, y)} J(t+1) \\
&\quad = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \frac{\nabla\omega(G_{t+1}|\phi, G_t, Y_t)}{\omega(G_{t+1}|\phi, G_t, Y_t)} J(t+1) \\
&\quad = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \frac{\nabla\omega(G_{t+1}|\phi, G_t, Y_t)}{\omega(G_{t+1}|\phi, G_t, Y_t)} \left[\sum_{s=t+1}^{T} \beta^{s-t-1} r(S_s, G_s) + \sum_{s=T+1}^{\infty} \beta^{s-t-1} r(S_s, G_s) \right].
\end{aligned} \tag{A.11}$$

Concentrating on the second term in the right-hand-side of (A.11) we observe that

$$\begin{aligned}
&\left\| \frac{1}{T} \sum_{t=0}^{T-1} \frac{\nabla\omega(G_{t+1}|\phi, G_t, Y_t)}{\omega(G_{t+1}|\phi, G_t, Y_t)} \sum_{s=T+1}^{\infty} \beta^{s-t-1} r(S_s, G_s) \right\| \\
&\leq \frac{1}{T} \sum_{t=0}^{T-1} \left\| \frac{\nabla\omega(G_{t+1}|\phi, G_t, Y_t)}{\omega(G_{t+1}|\phi, G_t, Y_t)} \right\| \sum_{s=T+1}^{\infty} \beta^{s-t-1} |r(S_s, G_s)|
\end{aligned}$$

$$
\begin{aligned}
&\leq \frac{BR}{T}\sum_{t=0}^{T-1}\sum_{s=T+1}^{\infty}\beta^{s-t-1}\\
&= \frac{BR}{T}\sum_{t=0}^{T-1}\frac{\beta^{T-t}}{1-\beta}\\
&= \frac{BR\beta\left(1-\beta^{T}\right)}{T\left(1-\beta\right)^{2}}\\
&\to 0\,\text{as}\; T\to\infty,
\end{aligned}
$$

where R and B are the bounds on the magnitudes of the rewards and $\|\nabla\omega/\omega\|$ from Assumptions 2 and 4 (see Section 3.2). Hence,

$$
\pi'(\nabla^{\phi}P)J_{\beta} = \lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\frac{\nabla\omega(G_{t+1}|\phi,G_t,Y_t)}{\omega(G_{t+1}|\phi,G_t,Y_t)}\sum_{s=t+1}^{T}\beta^{s-t-1}r(S_s,G_s), \tag{A.12}
$$

with probability 1. Unrolling the equation for Δ_T^{ϕ} in the IState-GPOMDP algorithm shows it is equal to

$$
\frac{1}{T}\sum_{t=0}^{T-1}\frac{\nabla\omega(g_{t+1}|\phi,g_t,y_t)}{\omega(g_{t+1}|\phi,g_t,y_t)}\sum_{s=t+1}^{T}\beta^{s-t-1}r_s,
$$

hence $\Delta_T^{\phi}\to\pi'\nabla^{\phi}PJ_{\beta}$ w.p.1 as required. The same procedure is followed to show that this is also true with respect to the parameters θ. □

A.3 Zero-Gradient Regions of FSC Agents

To increase η by making use of an FSC, we require $\|\nabla^{\phi}\eta\| > 0$. From Equation (4.8) we have

$$
\nabla\eta = \pi'\left[\sum_{n=0}^{\infty}(\nabla P)P^n\right]r. \tag{A.13}
$$

We must select (ϕ,θ) to provide an initial FSC and policy prior to training. A sensible choice for the FSC is one that makes the least assumptions about the task: a uniform FSC where any observation y is equally likely to lead to any next I-state h, from any current I-state g. Here we shall prove that this, and similarly sensible choices of initial FSC, result in $\|\nabla^{\phi}\eta\| = 0$.

A.3.1 Conditions for Zero Gradients

Recall from Equation (3.1) that the transition probability matrix $P(\phi, \theta)$ has dimension $|\mathcal{S}||\mathcal{G}| \times |\mathcal{S}||\mathcal{G}|$ and the entries $p(j, h|\phi, \theta, i, g)|_{i,j=1\ldots|\mathcal{S}|;g,h=1\ldots|\mathcal{G}|}$ are given by

$$p(j, h|\phi, \theta, i, g) = \sum_{y\in\mathcal{Y}} \sum_{u\in\mathcal{U}} \nu(y|i)\omega(h|\phi, g, y)\mu(u|\theta, h, y)q(j|i, u). \tag{A.14}$$

Also, $r = [r(1, 1), r(1, 2), \ldots , r(|\mathcal{S}|, |\mathcal{G}|)]'$ is a column vector of the rewards received for being in state (i, g). For FSCs $r(i, g) := r(i)$.

For the remainder of this section we will abuse notation to allow $\omega(h|\phi, g, y) = \omega(h|\phi, y)$ to mean $\omega(h|\phi, g, y) = \omega(h|\phi, g', y) \quad \forall g, g', y, h$ for the *specified choice of* ϕ *and* θ. In words, for these choices of ϕ and θ, $\omega(h|\phi, y)$ is independent of the choice of I-state g. When taking the gradient of these functions we will reintroduce the I-state dependence since the gradient may vary with the I-state even when the value does not.

Lemma 2. *If ϕ and θ are chosen such that $\omega(h|\phi, g, y) = \omega(h|\phi, y)$ $\forall g, h, y$ and $\mu(u|\theta, h, y) = \mu(u|\theta, y)$ $\forall h, u, y$, then $(\nabla^\phi P)P = [0]$.*

Proof. We start by re-writing Equation (3.1), taking into account the simplified distributions

$$p(j, h|\phi, \theta, i, g) = \sum_{y,u} \nu(y|i)\omega(h|\phi, y)\mu(u|\theta, y)q(j|i, u). \tag{A.15}$$

Similarly we can write down the simplified gradient of the l'th parameter of ϕ, denoted ϕ_l

$$\begin{aligned}\frac{\partial p(j, h|\phi, \theta, i, g)}{\partial \phi_l} &= \frac{\partial}{\partial \phi_l} \left[\sum_{y,u} \nu(y|i)\omega(h|\phi, y)\mu(u|\theta, y)q(j|i, u)\right] \\ &= \sum_{y,u} \nu(y|i) \frac{\partial \omega(h|\phi, g, y)}{\partial \phi_l} \mu(u|\theta, y)q(j|i, u).\end{aligned}$$

Now, the $(i, g)(j, h)$'th element of $(\frac{\partial P}{\partial \phi_l})P$ is the dot product of row (i, g) of $\frac{\partial P}{\partial \phi_l}$ with column (j,h) of P. Here $(k,\bar{c})$ defines which dot product element we are computing

$$\begin{aligned}\left(\frac{\partial P}{\partial \phi_l}P\right)_{(i,g)(j,h)} &= \sum_{k\in\mathcal{S}}\sum_{\bar{c}\in\mathcal{G}}\frac{\partial p(i,g|\theta,\phi,k,\bar{c})}{\partial \phi_l}p(j,h|\phi,\theta,k,\bar{c})\\ &= \sum_{k\in\mathcal{S}}\sum_{\bar{c}\in\mathcal{G}}\left(\sum_{y,u}\nu(y|i)\frac{\partial\omega(\bar{c}|\phi,g,y)}{\partial\phi_l}\mu(u|\theta,y)q(k|i,u)\right)\\ &\qquad\left(\sum_{y,u}\nu(y|i)\omega(h|\phi,y)\mu(u|\theta,y)q(j|k,u)\right).\end{aligned}$$

From Equation (A.15) we see that $p(j,h|\phi,\theta,k,\bar{c})$ does not depend on $\bar{c}$, so we define $p(j,h|\phi,\theta,k,\bar{c}) := p(j,h|\theta,\phi,k)$ and continue by moving the sum over $\bar{c}$ inside $\frac{\partial p(i,g|\theta,\phi,k,\bar{c})}{\partial\phi_l}$

$$\begin{aligned}&= \sum_{k\in\mathcal{S}}\left(\sum_{y,u}\nu(y|i)\mu(u|\theta,y)q(k|i,u)\sum_{c\in\mathcal{G}}\frac{\partial\omega(\bar{c}|\phi,g,y)}{\partial\phi_l}\right)p(j,h|\theta,\phi,k)\\ &= \sum_{k\in\mathcal{S}}\left(\sum_{y,u}\nu(y|i)\mu(u|\theta,y)q(k|i,u)\frac{\partial}{\partial\phi_l}\sum_{c\in\mathcal{G}}\omega(\bar{c}|\phi,g,y)\right)p(j,h|\theta,\phi,k)\\ &= \sum_{k\in\mathcal{S}}\left(\sum_{y,u}\nu(y|i)\mu(u|\theta,y)q(k|i,u)\frac{\partial}{\partial\phi_l}1\right)p(j,h|\theta,\phi,k)\\ &= \sum_{k\in\mathcal{S}}\left(\sum_{y,u}\nu(y|i)\mu(u|\theta,y)q(k|i,u)0\right)p(j,h|\theta,\phi,k)\\ &= [0].\end{aligned}$$

□

Lemma 3. *Under the same conditions as Lemma 2, $(\nabla^{\phi}P)r = 0$.*

Proof. The (i,g)'th element of $(\frac{\partial P}{\partial\phi_l})r$ is the dot product of row (i,g) of $\frac{\partial P}{\partial\phi_l}$ with r. By definition $r(l,\bar{c}) = r(l)$ is independent of $\bar{c}$, giving us

$$\left(\frac{\partial P}{\partial\phi_l}r\right)_{(i,g)} = \sum_{k\in\mathcal{S}}\sum_{\bar{c}\in\mathcal{G}}\left(\sum_{y,u}\nu(y|i)\frac{\partial\omega(\bar{c}|\phi,g,y)}{\partial\phi_l}\mu(u|\theta,y)q(k|i,u)\right)r(l).$$

Now we move the sum over $\bar{c}$ inside $\frac{\partial p(i,g|\theta,\phi,k,\bar{c})}{\partial\phi_l}$. The rest of the proof is the same as Lemma 2. □

Theorem 10. *If we choose θ and ϕ such that $\omega(h|\phi,g,y) = \omega(h|\phi,g',y)\ \forall h,g,g',y$ and $\mu(u|\theta,h,y) = \mu(u|\theta,h',y)\ \forall u,h,h',y$ then $\nabla^{\phi}\eta = [0]$.*

Proof. We expand Equation (A.13) for gradient element ϕ_l by taking out the first term of the summation, and applying Lemmas 2 and 3

$$\begin{aligned}\frac{\partial\eta}{\partial\phi_l} &= \pi'\left[\frac{\partial P}{\partial\phi_l} + \sum_{n=1}^{\infty}\frac{\partial P}{\partial\phi_l}P^n\right]r \\ &= \pi'\frac{\partial P}{\partial\phi_l}r + \pi'\left[\sum_{n=1}^{\infty}\frac{\partial P}{\partial\phi_l}P^n\right]r \\ &= \pi'[0] + \pi'\left[\sum_{n=1}^{\infty}\frac{\partial P}{\partial\phi_l}PP^{n-1}\right]r \\ &= \pi'\left[\sum_{n=1}^{\infty}[0]P^{n-1}\right]r \\ &= \pi'[0]r \\ &= 0.\end{aligned}$$

□

A.3.2 Conditions for Perpetual Zero Gradient

So far we have not shown any results concerning $\nabla^{\theta}\eta$. It is possible that $\|\nabla^{\theta}\eta\| > 0$ even under conditions for Theorem 10. There are at least three situations in which this can happen:

1. In some POMDPs it is possible to increase η by changing the stationary distribution of actions, that is, emitting some actions more frequently than others regardless of the observation. If this is possible then $\|\nabla^{\theta}\eta\| > 0$ is possible.

2. If the μ process is a function of y, then $\mu(u|\theta, h, y)$ can represent the optimal reactive policy, ignoring the I-state h.

3. If we choose ϕ such that $\omega(h|\phi, g, y) = \omega(h|\phi, y) \neq 1/|\mathcal{G}|$, then h tells us something about y even though it tells us nothing about the previous I-state, g. In this case $\omega(h|\phi, y)$ is a *partial observation hiding* process in the same way that $\nu(y|i)$ is a partial state hiding process. Because h is still a useful indicator of state, $\nabla^{\theta}\eta$ may have different values for parameters related to different I-states, indicating that internal state can help to maximise η. A single iteration of gradient ascent may therefore cause the condition $\mu(u|\theta, h, y) = \mu(u|\theta, y)$ of Theorem 10 to be violated, such that the next computation of the gradient results in $\|\nabla^{\phi}\eta\| > 0$.

The last case is interesting because it allows us to initialise $\mu(u|\theta, h, y) = \mu(u|\theta, y)$ and $\omega(h|\phi, g, y) = \omega(h|\phi, y)$ and possibly still learn to use I-states despite Theorem 10 telling us that these initialisations are sufficient for $\|\nabla^{\phi}\eta\| = 0$. We have verified that this can occur in practice for the Load/Unload scenario.

Whether it can happen depends on the choice of parameterisation for $\mu(u|\theta, h, y)$. We will now show that for $\mu(u|\theta, h, y)$ parameterised by a real-valued table and a soft-max output distribution (see Section 3.4.1), and under slightly tighter initialisation conditions than Theorem 10, that it is not possible for these conditions to be violated. Consequently, $\|\nabla^{\phi}\eta\| = 0$ from one gradient ascent iteration to another, resulting in perpetual failure to learn to use I-states.

Lemma 4. *Set ϕ such that $\omega(h|\phi, g, y) = 1/|\mathcal{G}|$ and choose some $\mu(u|\theta, h, y) = \mu(u|\theta, y)$ $\forall h, g, y, u$. Then $\nabla^{\theta}\eta$ is independent of the I-state.*

Proof. If we rewrite Equation (3.1) using the simplified definitions we find there is no dependence on the I-state so we can define $p(j, h|\phi, \theta, k, \bar{c}) := p(j|\phi, \theta, k)$. The gradient of Equation (3.1) w.r.t. the ϕ parameters is

$$\begin{aligned}\frac{\partial p(j, h|\phi, \theta, i, g)}{\partial \theta_c} &= \frac{\partial}{\partial \theta_c}\left[\sum_{y,u} \nu(y|i)\frac{1}{|\mathcal{G}|}\frac{\partial \mu(u|\theta, h, y)}{\partial \theta_c} q(j|i, u)\right] \\ &= \frac{1}{|\mathcal{G}|}\sum_{y,u} \nu(y|i)\frac{\partial \mu(u|\theta, h, y)}{\partial \theta_c} q(j|i, u).\end{aligned}$$

Now even if $\nabla^{\theta}\mu(u|\theta, h, y)$ is generally a function of h, $\nabla^{\theta}\eta$ is independent of h under the conditions given. This is because the dependence of the gradient w.r.t. any parameter θ_c on the I-state h is always marginalised out during the dot-product

$$\begin{aligned}\left(\frac{\partial P}{\partial \theta_c}P\right)_{(i,g)(j,h)} &= \sum_{k\in\mathcal{S}}\sum_{\bar{c}\in\mathcal{G}}\left(\frac{1}{|\mathcal{G}|}\sum_{u,y}\nu(y|i)q(k|i, u)\frac{\partial \mu(u|\theta, \bar{c}, y)}{\partial \theta_c}\right)p(j|\phi, \theta, k) \\ &= \sum_{k\in\mathcal{S}}\left(\frac{1}{|\mathcal{G}|}\sum_{u,y}\nu(y|i)q(k|i, u)\sum_{\bar{c}\in\mathcal{G}}\frac{\partial \mu(u|\theta, \bar{c}, y)}{\partial \theta_c}\right)p(j|\phi, \theta, k)\end{aligned}$$

At this point we observe that $(\frac{\partial P}{\partial \theta_c})P$ is completely independent of the internal state $\forall c$. Because the rewards are independent of the internal state it follows that $(\frac{\partial P}{\partial \theta_c})r$ is also independent of the internal state. Furthermore, if we can show that π is independent of the internal state then it follows from Equation (A.13) that $\nabla^{\theta}\eta$ is independent of the internal state.

Since we have defined $\omega(h|\phi, g, y) = 1/|\mathcal{G}|$, and with the further assumption that the *initial* distribution on I-states is $\pi(g) = 1/|\mathcal{G}|$, then the probability of being in any I-state at any time is $1/|\mathcal{G}|$. This allows us to state $\pi(i, g) = \frac{\pi(i)}{|\mathcal{G}|}$. Therefore, all components of Equation (A.13) are independent of the current internal state and thus $\nabla^{\theta}\eta$ is independent of the internal state.

□

After estimating the gradient we make a step $\Delta\theta := \gamma\nabla^{\theta}\eta$, where γ is a positive step size. Whether or not the conditions of Lemma 4 lead to the perpetual 0 gradient situation depends on whether $\mu(u|\theta + \Delta\theta, h, y)$ alters the conditions for Lemma 4. These conditions can be broken even when $\nabla^{\theta}\eta$ is independent of the I-state in the way Lemma 4 defines.

So we need to prove $\mu(u|\theta + \Delta\theta, h, y) = \mu(u|\theta + \Delta\theta, y)$ for all h for the parameterisation of $\mu(u|\theta, h, y)$ that we are interested in. We start by showing this is true for a lookup table indexed by (g, y) and that provides an $\mathbb{R}^{|\mathcal{U}|}$ vector which is turned into an action distribution using a soft-max function (3.8) (equivalently a Boltzmann function with a temperature co-efficient of 1.0). Since the use of the soft-max distribution is a common choice for translating real number vectors into distributions, the following proof forms a basis for showing many common choices of $\mu(u|\theta, h, y)$ can lead to perpetual $\|\nabla^{\phi}\eta = 0\|$ situations.

Lemma 5. *Let $x, y \in \mathbb{R}^{|\mathcal{U}|}$ and $u \in \mathcal{U}$ where $\mathcal{U} = \{1, \ldots, |\mathcal{U}|\}$. Define $d(u) : \mathcal{U} \mapsto \mathbb{R}$. Assume*

$$\frac{\exp(x_u)}{\sum_{u' \in \mathcal{U}} \exp(x_{u'})} = \frac{\exp(y_u)}{\sum_{u' \in \mathcal{U}} \exp(y_{u'})} \quad \forall u, \tag{A.16}$$

then

$$\frac{\exp(x_u + d(u))}{\sum_{u' \in \mathcal{U}} \exp(x_{u'} + d(u'))} = \frac{\exp(y_u + d(u))}{\sum_{u' \in \mathcal{U}} \exp(y_{u'} + d(u'))} \quad \forall u. \tag{A.17}$$

Proof. For (A.16) to be true, the x_u's and y_u's must differ by at most some constant c_u. This can be shown with a short proof by contradiction starting with $x_u = y_u + c_u$ and assuming $\exists c_{u'} \neq c_u$ for which the equality holds. This fact and some algebra give us the result

$$\begin{aligned}
\frac{\exp(x_u + d(u))}{\sum_{u'} \exp(x'_u + d(u'))} &= \frac{\exp(c)\exp(x_u + d(u))}{\sum_{u'} \exp(c)\exp(x'_u + d(u'))} \\
&= \frac{\exp(x_u + c + d(u))}{\sum_{u'} \exp(x'_u + c + d(u'))} \\
&= \frac{\exp(y_u + d(u))}{\sum_{u'} \exp(y'_u + d(u'))}
\end{aligned}$$

□

This lemma tells us that if we have two equal soft-max distributions generated by the possibly different vectors x and y, then adding a quantity independent of x or y (but possibly dependent on the element index) to both vectors results in distributions that remain equal. This is useful because we wish to show that when we have two vectors generated by different I-states, which result in the same output distribution, then adding a quantity independent of the I-state does not change the equality of the distributions. Also unchanged is the independence of the distributions with respect to the I-state conditioning.

Lemma 6. *For $\mu(u|\theta, h, y)$ parameterised by a lookup-table with soft-max output distribution, and under the conditions of Lemma 4, then $\mu(u|\theta+\Delta\theta, h, y) = \mu(u|\theta+\Delta\theta, y)$ implies $\|\nabla^{\phi}\eta\| = 0$ always.*

Proof. Let $x_u = \theta_{uhy}$ and $y_u = \theta_{u\bar{h}y}$. If we set

$$d(u) = \gamma \frac{\partial \eta}{\partial \theta_{uhy}},$$

then the *left* hand side of (A.17) is equal to $\mu(u|\theta + \Delta\theta, h, y)$. If we set

$$d(u) = \gamma \frac{\partial \eta}{\partial \theta_{u\bar{h}y}},$$

then the *right* hand side gives us $\mu(u|\theta + \Delta\theta, \bar{h}, y)$. However, Lemma 5 requires $d(u)$ be the same on the left and right sides, which is precisely what Lemma 4 tells us is the case, so

$$d(u) = \gamma \frac{\partial \eta}{\partial \theta_{uhy}} = \gamma \frac{\partial \eta}{\partial \theta_{u\bar{h}y}}.$$

Lemma 5 can be applied for all choices of h and $\bar{h}$, resulting in

$$\begin{aligned}\mu(u|\theta + \Delta\theta, h, y) &= \mu(u|\theta + \Delta\theta, \bar{h}, y) \quad \forall h, \bar{h} \\ &= \mu(u|\theta + \Delta\theta, y).\end{aligned}$$

□

In summary, Lemma 6 tells us that if we meet the necessary conditions for Lemma 4, and we parameterise $\mu(u|\theta, h, y)$ using a lookup-table and the soft-max function, then $\|\nabla^{\phi}\eta\| = 0$ from one step of the IState-GPOMDP algorithm to another. Thus, by induction, we have $\|\nabla^{\phi}\eta\| = 0$ always.

Because we have based the proof on the value of the true gradient, the result holds for any algorithm that estimates the gradient. This includes IState-GPOMDP and I-state Williams' REINFORCE [Peshkin et al., 1999].

The analysis above presents a method for avoiding 0 gradient regions of parameter space. We simply select initial controllers that violate the conditions of Theorem 10. By making the set of future states reachable from I-state g a subset of $\mathcal{G}$, and by ensuring that the subset we can reach is different for all g and y, we neatly avoid the problem. At the same time, using subsets increases computational efficiency by introducing sparseness. This allows us to use large I-state spaces without slowing down each step of the computation. However, large I-state spaces will require more steps.

Lemma 4 assumes a lookup-table controller with a soft-max output distribution. The following generalises the same argument to arbitrary policies with soft-max output distributions.

Theorem 11. *Let* $f(\theta, h, y, u) : \mathbb{R}^{n_\theta} \times \mathcal{G} \times \mathcal{Y} \times \mathcal{U} \mapsto \mathbb{R}$. *Then using a soft-max output distribution we can write*

$$\mu(u|\theta, h, y) = \frac{\exp(f(\theta, h, y, u))}{\sum_{u' \in \mathcal{U}} \exp(f(\theta, h, y, u'))}.$$

If $\mu(u|\theta, h, y)$ *fulfills the conditions of Lemma 4, and if we can write* $\mu(u|\theta, h, y)$ *after a step in the gradient direction as*

$$\mu(u|\theta, h, y) = \frac{\exp(f(\theta, h, y, u) + f(\Delta\theta, y, u))}{\sum_{u' \in \mathcal{U}} \exp(f(\phi, h, y, u') + f(\Delta\theta, y, u'))},$$

then $\|\nabla^\phi \eta\| = 0$ *always.*

Proof. The theorem is a generalisation of Lemma 6. By allowing an arbitrary function $f(\theta, h, y, u)$ in place of θ_{uhy}, and if a parameter step of $\Delta\theta$ is separable so that

$$f(\theta + \Delta\theta, h, y, u) = f(\theta, h, y, u) + f(\Delta\theta, y, u),$$

then the same argument as Lemma 6 holds. We require the second term to be independent of h so that it takes the place of $d(u)$ in Lemma 6. □

Using Theorem 11 we can easily show that 0 gradient problems exist for linear controllers

$$f(\theta, h, y, u) = \sum_{c=1}^{n_f} \psi_c(h, y)\theta_{cu}.$$

that is, a controller with n_f features $\{\psi_1(h, y), \ldots, \psi_{n_f}(h, y)\}$ and parameters θ_{cu} where $c = 1, \ldots, n_f; u = 1, \ldots, |\mathcal{U}|$. Linear controllers are separable as required by the corollary.

One simple choice of $|\mathcal{Y}| + n_f + 1$ features we used in early experiments results is

$$f(\theta, h, y, u) = \theta_{hu} + \theta_{yu} + \theta_u,$$

and the change in value to $f(\theta, h, y, u)$ after a step is

$$f(\Delta\theta, h, y, u) = \gamma \left(\frac{\partial \eta}{\theta_{hu}} + \frac{\partial \eta}{\theta_{yu}} + \frac{\partial \eta}{\theta_{u}} \right).$$

Lemma 4 tells us this is equal to $f(\Delta\theta, h', y, u)$ for all h, h' and appropriate choice of initial θ. So all the conditions of Theorem 11 are satisfied with appropriate choices of initial θ and ϕ, implying $\|\nabla^{\phi}\eta\| = 0$ always.

We can also show a simple example that demonstrates that not all linear controllers are subject to the 0 gradient regions of parameter space we have analysed.[1] Let there be only 1 feature $\psi_1(h, y) = h$. If we set $\omega(h|\phi, g, y) = 1/|\mathcal{G}|$ and $\theta = 0$ then all the conditions of Theorem 10 are met. However, the change in $f(\theta, h, y, u)$ is

$$f(\theta_{1u}, h, y, u) = h\frac{\partial \eta}{\theta_{1u}}$$

which is not independent of h as required by Theorem 11.

As a final linear example, we retrieve Theorem 6 from Corollary 11 by defining $n_f = |\mathcal{G}||\mathcal{Y}|$ input features:

$$\psi_{\bar{h}\bar{y}}(h, y) = \chi_{\bar{h}}(h)\chi_{\bar{y}}(y),$$

which is simply a binary input for every possible combination of observation and internal state.

A.4 FIR Trace Filter Proofs

This section provides proofs of the theorems stated in Section 9.1.

A.4.1 Proof of Theorem 8

The following proof is brief at it is a slightly modified version of the proof of Theorem 5 in Appendix A.2.1.

Proof. The filtered reward is defined as

$$J_f(\phi, \theta, i, g) := \mathbb{E}_{\phi,\theta} \left[\sum_{n=0}^{|b|-1} b^n r(i_n, g_n) | i_0 = i, g_0 = g \right],$$

[1]Avoiding the conditions of Theorem 10 does not mean we will avoid all 0 gradient regions. The use of the soft-max function means the gradient approaches 0 in the limit as $f(\theta, h, y, u) \to \infty$. Other 0 gradient situations may exist and are a well known problem of gradient optimisation.

where the expectation is over all world/I-state trajectories induced by the parameter settings. The filter is defined by the vector of *taps* b, where the $n'th$ tap is given by b_{n-1}, so that the first element of b is given index 0. The number of taps $|b|$ is possibly infinite so any FIR or IIR filter can be represented by J_f. We will now simplify the notation by letting the state i index the global state, the cross product of the world state and I-state. Having an explicit I-state and global state will not change the result. We may then write

$$\begin{aligned}
J_f(i) = b_0 r(i_0) &+ b_1 \sum_{i_1 \in \mathcal{S}} P(i_1|\theta,\phi,i_0) r(i_1) \\
&+ b_2 \sum_{i_1} \sum_{i_2} P(i_1|\theta,\phi,i_0) P(i_2|\phi,\theta,i_1) r(i_2) \\
&\vdots \\
&+ b_{|b|-1} \sum_{i_1} \cdots \sum_{i_{|b|-1}} P(i_1|\phi,\theta,i_0) \cdots P(i_{|b|-1}|\phi,\theta,i_{|b|-2}) r(i_{|b|-1}),
\end{aligned}$$

where P is defined by Equation (3.1). This can be neatly re-written in matrix form where J_f is now a vector of length $|\mathcal{S}|$ representing the filtered reward starting from each state i. As usual, we also make the dependencies on ϕ and θ implicit, obtaining

$$J_f = \left[\sum_{n=0}^{|b|-1} b_n P^n \right] r. \tag{A.18}$$

From the balance equations $\pi' P = \pi'$ we have

$$\begin{aligned}
\nabla \pi' P &= \nabla \pi' \\
(\nabla \pi')P + \pi'(\nabla P) &= \nabla \pi' \\
(\nabla \pi')(I - P) &= \pi'(\nabla P).
\end{aligned}$$

Recall from Section 4.1 and particularly Equation (4.3), that the matrix $(I - P)$ is not invertible, so we condition it using $(I - P + e\pi')$ where e is a column vector of 1's of length $|\mathcal{S}||\mathcal{G}|$. This gives

$$(\nabla \pi') = \pi'(\nabla P)(I - P + e\pi')^{-1},$$

which we substitute into the gradient of the long-term average reward $\nabla \eta = (\nabla \pi')r$ to obtain

$$\begin{aligned}\nabla\eta &= \pi'(\nabla P)(I - P + e\pi)^{-1}r \\ &= \pi'(\nabla P)\sum_{n=0}^{\infty}(P - e\pi)^n r \\ &= \pi'(\nabla P)\sum_{n=0}^{\infty} P^n r, \qquad \text{from (4.8).} \end{aligned} \tag{A.19}$$

Since we have assumed $|b| \to \infty$ and $b_n = 1,\ \forall n = 0, \ldots, \infty$, we immediately observe that (A.18) is the same as the summation in (A.19), giving us the desired result

$$(\nabla\pi') = \pi'(\nabla P)J_f.$$

$\square$

A.4.2 Proof of Theorem 7

The proof is essentially the same as the proof of Theorem 4, given in more detail in Appendix A.2.2.

Proof. We replace J_β with J_f and thus $J(t+1)$ is the process governing the filtered reward

$$J(t+1) = \sum_{s=t+1}^{t+|b|} b_{s-t-1}r(S_s, G_s),$$

such that

$$J_f(j,h) = \mathbb{E}[J(t+1)|S_{t+1} = j, G_{t+1} = h]$$

follows from the boundedness of the rewards and Lebesgue's dominated convergence theorem.

Because $J(t+1)$ is a finite sum we avoid having the further approximation introduced by eliminating the second term in Equation (A.11). In place of (A.11) we have for the gradient w.r.t. ϕ

$$\pi'(\nabla^\phi P)J_f = \lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\frac{\nabla\omega(G_{t+1}|\phi, G_t, Y_t)}{\omega(G_{t+1}|\phi, G_t, Y_t)}\sum_{s=t+1}^{t+|b|} b_{s-t-1}r(S_s, G_s),$$

with probability one. Now replacing line 7 of Algorithm 2 with

$$z_{t+1} = \sum_{n=0}^{|b|-1} b_n \frac{\nabla\omega(g_{t+1-n}|\phi, g_{t-n}, y_{t-n})}{\omega(g_{t+1-n}|\phi, g_{t-n}, y_{t-n})},$$

and unrolling the equation for Δ_T^ϕ of the modified algorithm shows it is equal to

$$\frac{1}{T}\sum_{t=0}^{T-1}\frac{\nabla\omega(g_{t+1}|\phi,g_t,y_t)}{\omega(g_{t+1}|\phi,g_t,y_t)}\sum_{s=t+1}^{t+|b|}b_{s-t-1}r(i_s,g_s),$$

hence $\Delta_T^\phi \to \pi'(\nabla^\phi P)J_f$ w.p.1 as required. The same procedure is followed to show that this is also true with respect to the parameters θ. □

The IIR style filter given by the trace update of Equation (9.1) is an efficient implementation of infinite versions of the FIR filters presented in this proof, allowing a finite number of filter taps to produce infinite impulse responses. The IIR update does not expand the space of filters covered by this proof.

A.5 Proof of Theorem 9

The following proof of Theorem 9 demonstrates that importance sampling cannot be immediately applied to IState-GPOMDP in the style used by Meuleau et al. [2000].

Proof. We begin by summarising the first part of the proof of Theorem 4 (see Appendix A.2.2). To make the proof clearer we do not include the FSC $\omega(h|\phi,g,y)$, that is, we assume memory-less IState-GPOMDP. Thus, the proof starts in the same way as the original proof of the GPOMDP algorithm [Baxter and Bartlett, 2001]

$$\begin{aligned}
\pi'(\nabla P)J_\beta &= \sum_{i,j} J_\beta(j)\pi(i|\theta)\nabla p(j|\theta,i)\\
&= \sum_{i,j} J_\beta(j)\pi(i|\theta)\nabla\sum_{y,u}\nu(y|i)\mu(u|\theta,y)q(j|i,u) \quad \text{, from (3.1)}\\
&= \sum_{i,j,y,u} J_\beta(j)\pi(i|\theta)\nu(y|i)\nabla\mu(u|\theta,y)q(j|i,u)\\
&= \sum_{i,j,y,u} J_\beta(j)\frac{\nabla\mu(u|\theta,y)}{\mu(u|\theta,y)}\mu(u|\theta,y)\pi(i|\theta)\nu(y|i)q(j|i,u)\\
&= \mathbb{E}_{S_t,Y_t,U_t,S_{t+1}}[J_\beta(S_{t+1})\nabla\ln\mu(U_t|\theta,Y_t)] \qquad \text{(A.20)}
\end{aligned}$$

So far nothing has changed from the IState-GPOMDP proof (except for the absence of the memory represented by $\omega(h|\phi,g,y)$). Now we introduce importance sampling. Suppose $w(X)$ is the p.d.f. for the discrete distribution of the random variable X and $s(X')$ is the *known* p.d.f. for the discrete distribution of the random variable X'. Importance sampling tells us [Glynn, 1996] that we can compute the expectation of

$f(X)\nabla \ln w(X)$ by sampling X'

$$\begin{aligned}\mathbb{E}_X[f(x)\nabla \ln w(x)] &= \sum_{x\in X} f(x)(\nabla \ln w(x))w(x) \\ &= \sum_{x\in X} f(x)(\nabla \ln w(x))\frac{w(x)}{s(x)}s(x) \\ &= \mathbb{E}_{X'}\left[f(x)(\nabla \ln w(x))\frac{w(x)}{s(x)}\right].\end{aligned} \tag{A.21}$$

The result is that we can estimate the expectation according to the distribution X by sampling from the distribution X'. Now we apply this principle to Equation (A.20) using $w(i,y,u,j) = \pi(i|\theta)\nu(y|i)\mu(u|\theta,y)q(j|i,u)$, the probability of moving from state i to j after observation y and choosing action u, and $s(i,y,u,j) = \tilde{\pi}(i)\nu(y|i)\tilde{\mu}(u|y)q(j|i,u)$, which is the probability under the distribution set up by the teacher. Starting with Equation (A.20) and following exactly the same procedure as we did for Equation (A.21), we obtain

$$\begin{aligned}\pi'(\nabla P)J_\beta &= \sum_{i,y,u,j} J_\beta(j)\frac{\nabla\mu(u|\theta,y)}{\mu(u|\theta,y)}\frac{\pi(i|\theta)\nu(y|i)\mu(u|\theta,y)q(j|i,u)}{\tilde{\pi}(i)\nu(y|i)\tilde{\mu}(u|y)q(j|i,u)}\tilde{\pi}(i)\nu(y|i)\tilde{\mu}(u|y)q(j|i,u) \\ &= \sum_{i,y,u,j} J_\beta(j)(\nabla\mu(u|\theta,y))\frac{\pi(i|\theta)}{\tilde{\pi}(i)}\frac{1}{\tilde{\mu}(u|y)}\tilde{\pi}(i)\nu(y|i)\tilde{\mu}(u|y)q(j|i,u)\end{aligned}$$

$$= \sum_{i,y,u,j} J_\beta(j)\frac{\nabla\mu(u|\theta,y)}{\tilde{\mu}(u|y)}\frac{\pi(i|\theta)}{\tilde{\pi}(i)}\tilde{\pi}(i)\nu(y|i)\tilde{\mu}(u|y)q(j|i,u) \tag{A.22}$$

$$= \sum_{i,y,u,j} \mathbb{E}_{\phi,\theta}\chi_i(\tilde{S}_t)\chi_y(\tilde{Y}_t)\chi_u(\tilde{U}_t)\chi_j(\tilde{S}_{t+1})J(t+1)\frac{\nabla\mu(U_t|\theta,Y_t)}{\tilde{\mu}(U_t|Y_t)}\frac{\pi(S_t|\theta)}{\tilde{\pi}(S_t)} \tag{A.23}$$

The rest of the proof follows through as it does in Appendix A.2.2. The critical point is that we failed to eliminate references to the stationary distribution as we did in the IState-GPOMDP proof, necessitating some method for estimating the ratio $\frac{\pi(i|\theta)}{\tilde{\pi}(i)}$. The same situation holds true when memory is added in the form of an FSC, with the slight complication that the stationary distribution must also be calculated over the I-states.

□

Appendix B

Gradient Ascent Methods

This appendix provides details of the gradient ascent method we use to maximise the goal function of all the gradient based algorithms in this thesis, including the conjugate-gradient algorithm, the GSEARCH algorithm, and the use of quadratic penalties.

B.1 The GSEARCH Algorithm

Once a gradient estimate has been obtained the conjugate-gradient algorithm ConjGrad [Fine, 1999, §5.5.2] computes the search direction θ^*. For brevity, this Appendix assumes that θ is the set of parameters for the entire system, that is, it encompasses both the ϕ and θ parameters used throughout the thesis. The conjugate-gradient algorithm (Algorithm 5) ensures that successive search directions are orthogonal. This helps avoid problems such as successive parameter updates oscillating between opposite walls of a ridge in η. It is preferable to move in an uphill direction along the ridge top.

The parameters are updated using $\theta_{t+1} = \theta_t + \gamma\theta^*$ where γ is a positive step size. A line search is often used to chose γ, trying a sequence of exponentially increasing step sizes $\{\gamma_1, \gamma_2, \dots\}$, attempting to bracket the maximum *value* of η in the search direction. After bracketing the maximum, quadratic interpolation is used to compute a final value for γ. Unfortunately, our Monte-Carlo estimates of η for each γ_s are noisy. This can result in the failure to detect the true maximum in the search direction.

A more robust line search procedure by Baxter et al. [2001a] is GSEARCH. This procedure is given by Algorithm 6. Instead of estimating the value η at each γ_s, we compute local gradient estimates $\nabla\eta(\theta_t + \theta^*\gamma_s)$. Let ϵ be the machine tolerance for 0. While

$$\operatorname{sgn}(\nabla\eta(\theta_t + \theta^*\gamma_s) \cdot \theta^*) > \epsilon,$$

we have not yet bracketed the maximum. This equation checks that the local gradient and the search direction θ^* are within 90° of each other, and hence point in roughly the same direction. Once they are further than 90° apart, the sign of the dot product

Algorithm 5 ConjGrad

```
1: Given:
     • $\nabla\colon \mathbb{R}^{n_\phi+n_\theta} \to \mathbb{R}^{n_\phi+n_\theta}$: a (possibly noisy and biased) estimate of the gradient
       of the objective function to be maximised.
     • Starting parameters $\theta \in \mathbb{R}^{n_\phi+n_\theta}$ (set to maximum on return).
     • Initial step size $\gamma_0 > 0$.
     • Gradient resolution $\epsilon$.
2: $g = \theta^* = \nabla(\theta)$
3: while $\|g\|^2 \geq \epsilon$ do
4:   $\theta \leftarrow \mathsf{GSEARCH}(\nabla, \theta, \theta^*, \gamma_0, \epsilon)$
5:   $\Delta = \nabla(\theta)$
6:   $\psi = (\Delta - g) \cdot \Delta / \|g\|^2$
7:   $\theta^* = \Delta + \psi\theta^*$
8:   if $\theta^* \cdot \Delta < 0$ then
9:     $\theta^* = \Delta$
10:  end if
11:  $g = \Delta$
12: end while
13: return $\theta$
```

changes, indicating that the local gradient estimate is pointing downhill (assuming the noisy search direction pointed uphill). Thus, all we need to do is check for a flip in the sign of the dot product which indicates that we have just stepped past a maximum.

To understand why GSEARCH is more robust than value bracketing, begin by picturing a plot of *noisy* estimates of a quadratic region of $\eta(\theta)$ around the true maximum. Also, picture a plot of *noisy* estimates of the linear gradient $\nabla\eta(\theta)$. An error in value bracketing could occur anywhere along the plot of the noisy value function, but errors using the sign of the gradient can only occur where zero-crossings of the gradient plot occur, which are only near the true maximum. GSEARCH can also be understood by noting that if the gradient magnitude is stronger than the noise, the noise has little effect; but noise in the value function is bad regardless of the value magnitude. It is possible to break such arguments, for instance, $\nabla\eta(\theta)$ estimates could be noisier than $\eta(\theta)$ estimates. However, in practice GSEARCH usually performs better than using value bracketing. A more detailed justification, including the plots we mentioned, is given by Baxter et al. [2001a].

After finding two points γ_s and γ_{s+1} that bracket a maximum, we perform quadratic interpolation between the two points to try and find the optimal step size γ. The quadratic interpolation is given by lines 28–31 of Algorithm 6, which appears linear because p_- and p_+ are dot products of the search direction θ^* with the local gradient $\nabla(\theta)$. This interpolation scheme assumes that at the local level we are solving a quadratic optimisation problem, similar to applying a step of Newton's method.

Because GSEARCH is reasonably robust, we can tolerate a poor local gradient estimate of $\nabla\eta(\theta_t + \theta^*\gamma_s)$. This allows the search to be performed with fewer simulation steps than a value based search. Empirically we found that the sign calculation is reliable using as few as a tenth of the number of steps used to estimate $\nabla\eta(\theta_t)$.

A useful check at the start of each line search is to ensure

$$\operatorname{sgn}(\nabla\eta(\theta_t) \cdot \theta^*) > \epsilon,$$

otherwise the search direction is pointing downhill compared to the local gradient estimate. If this check fails after one or more iterations of the line search it means that θ^* may need to be reset to last full estimate of $\nabla\eta$. If the check fails on the first iteration, or after setting $\theta^* = \nabla\eta$, it indicates that the gradient estimates have high variance. If this occurs the discount factor β and the estimation time T need to be re-tuned.

When the estimated gradients drop below a threshold ϵ, or the line search fails twice in a row, we declare an end to training.

B.2 Quadratic Penalties

Gradient estimates early in learning often point to a poor local maximum. Because the line search attempts to maximise $\eta(\theta_t+\gamma\theta^*)$ in one iteration, it is easy to end up caught in such a local maximum. Furthermore, for output distribution functions such as the soft-max function (3.8) the gradient tends to zero as the magnitude of the parameters tends to infinity, so if the line search steps too far in *any* direction, we will saturate the soft-max function and gradient ascent will terminate with a small gradient. We can avoid these local maxima by adding a constraint term to the search direction that penalises large parameter values. The constraint keeps decision boundaries close to the origin in parameter space. Intuitively, the penalty slows down the gradient ascent, giving it a chance to recover from errors.

For example, in the multi-agent problem of Section 4.5 the initial gradient direction tells the agents to move forward when the sensors fail. Without a penalty term the line search pushes the parameters for this situation to values that result in near 0 gradients during subsequent $\nabla\eta$ estimations. The end result is that the agents never learn the optimal behaviour, which is to wait when their sensors fail. However, if we restrict the initial parameter growth, the agents have a chance to correct their policies.

Algorithm 6 GSEARCH

```
1: **Given:**
   • $\nabla\colon \mathbb{R}^{n_\phi+n_\theta} \to \mathbb{R}^{n_\phi+n_\theta}$: a (possibly noisy and biased) estimate of the gradient of the objective function.
   • Starting parameters $\theta_0 \in \mathbb{R}^{n_\phi+n_\theta}$ (set to maximum on return).
   • Search direction $\theta^* \in \mathbb{R}^{n_\phi+n_\theta}$ with $\nabla(\theta_0) \cdot \nabla(\theta^*) > 0$.
   • Initial step size $\gamma_0 > 0$.
   • Inner product resolution $\epsilon >= 0$.
2: $\gamma = \gamma_0$
3: $\theta = \theta_0 + \gamma\theta^*$
4: $\Delta = \nabla(\theta)$
5: **if** $\Delta \cdot \theta^* < 0$ **then**
6:    Step back to bracket the maximum:
7:    **repeat**
8:       $\gamma_+ = \gamma$
9:       $p_+ = \Delta \cdot \theta^*$
10:      $\gamma = \gamma/2$
11:      $\theta = \theta_0 + \gamma\theta^*$
12:      $\Delta = \nabla(\theta)$
13:   **until** $\Delta \cdot \theta^* > -\epsilon$
14:   $\gamma_- = \gamma$
15:   $p_- = \Delta \cdot \theta^*$
16: **else**
17:   Step forward to bracket the maximum:
18:   **repeat**
19:      $\gamma_- = \gamma$
20:      $p_- = \Delta \cdot \theta^*$
21:      $\gamma = 2\gamma$
22:      $\theta = \theta_0 + \gamma\theta^*$
23:      $\Delta = \nabla(\theta)$
24:   **until** $\Delta \cdot \theta^* < \epsilon$
25:   $\gamma_+ = \gamma$
26:   $p_+ = \Delta \cdot \theta^*$
27: **end if**
28: **if** $p_- > 0$ and $p_+ < 0$ **then**
29:   $\gamma = \gamma_- - p_- \frac{\gamma_+ - \gamma_-}{p_+ - p_-}$
30: **else**
31:   $\gamma = \frac{\gamma_- + \gamma_+}{2}$
32: **end if**
33: return $\theta_0 = \theta_0 + \gamma\theta^*$
```

An alternative intuition arises by observing that large parameter values result in near deterministic policies, reducing exploration during simulation. If an agent does not experience the preferred policy, it cannot learn it.

We use a *quadratic* penalty term. Using quadratic penalties to meet inequality constraints during stochastic optimisation is covered in detail in Kushner and Clark [1978, §5.2]. Let $\vec{\wp} \in [0, \infty)^{n_\theta}$ be a vector of penalties, with $\wp_c$ giving the penalty for parameter θ_c. We define the penalised η as

$$\begin{aligned}\bar{\eta} &:= \eta - \sum_{c=1}^{n_\theta} \frac{\wp_c \theta_c^2}{2} \\ \frac{\partial \bar{\eta}}{\theta_c} &= \frac{\partial \eta}{\theta_c} - \wp_c \theta_c.\end{aligned}$$

In practice we use the same scalar penalty $\wp$ for all parameters, though in some situations, such as having independent sets of parameters θ and ϕ, non-uniform $\vec{\wp}$ could aid convergence.

To eventually settle into a maximum we reduce the penalty over time. We halve the penalty if $\bar{\eta}$ fails to increase by more than 2% over 3 or more iterations of GSEARCH.

Appendix C

Variance Reduction Continued

This appendix contains details of the IGPOMDP-SARSA algorithm along with preliminary experiments. It also provides details of why using fixed random number sequences can introduce over-fitting effects.

C.1 GPOMDP-SARSA Hybrid Details

In this section we briefly investigate using SARSA(λ) to learn the policy $\mu(u|\theta, h, y)$ and IState-GPOMDP to learn the FSC $\omega(h|\phi, g, y)$. This idea is motivated by two observations. Firstly, $\omega(h|\phi, g, y)$ can be considered a small and *fully observable* MDP, rendering it feasible to apply value based algorithms such as SARSA(λ) [Sutton and Barto, 1998]. Secondly, if our value based approach learns a successful policy before the FSC has converged, the improved actions will cause more relevant areas of the world-state space to be explored, improving the FSC more rapidly than if a trajectory based on actions generated by an inferior $\mu(u|\theta, h, y)$ was used. Shorter, more relevant, trajectories ease the difficulty of temporal credit assignment. The improved FSC then helps the value-function method to improve its estimates for the value of each state of the FSC, and so on.

The key idea behind both motivations is that value methods can sometimes learn in fewer steps than policy-gradient methods due to their lower variance, as discussed in Section 2.5.8. We chose to use SARSA(λ) because of its good empirical results on partially observable domains [Loch and Singh, 1998]. If we restrict ourselves to lookup-table representations for the value function, and $\lambda = 0$, SARSA(λ) will converge to a globally optimal policy given a fixed FSC [Sutton, 1999].

C.1.1 The SARSA(λ) Algorithm

Algorithm 7 is SARSA(λ) as presented in Sutton and Barto [1998]. It is an *on-policy* method for learning Q-functions using temporal differences. It is similar to the well known TD(λ) algorithm.

Algorithm 7 SARSA(λ).

```
1: Given:
     • Fixed Policy $\mu(\cdot|\theta, i)$.
     • $\Gamma, \lambda \in [0, 1)$ ($\lambda$ may be 1 for episodic tasks).
     • Initial parameter values $\theta \in \mathbb{R}^{n_\theta}$.
     • Differentiable Q-function class $\{Q(\theta, \cdot, \cdot) \colon \theta \in \mathbb{R}^{n_\theta}\}$.
     • Step sizes $\gamma_t, t = 0, 1, \dots$ satisfying $\gamma_t \geq 0$, $\sum \gamma_t = \infty$ and $\sum \gamma_t^2 < \infty$.
     • Arbitrary starting state $i_0$ and initial action $u_0$ sampled from $\mu(\cdot|\theta, i_0)$.
     • $z_0 = \nabla Q(\theta, i_0, u_0)$.
2: for each transition $i_t, u_t \to i_{t+1}, u_{t+1}$ generated according to the MDP and $\mu(\cdot|\theta, i_{t+1})$ do
3:   Compute $d(i_t, i_{t+1}) = r(i_t) + \Gamma Q(\theta_t, i_{t+1}, u_{t+1}) - Q(\theta_t, i_t, u_t)$
4:   Set $\theta_{t+1} = \theta_t + \gamma_t d(i_t, i_{t+1}) z_t$
5:   Set $z_{t+1} = \Gamma \lambda z_t + \nabla Q(\theta_{t+1}, i_{t+1}, u_{t+1})$
6: end for
```

When following a fixed policy, SARSA(λ) is guaranteed to converge for lookup-table and linear policy representations when $\lambda = 0$. If function approximation is introduced then we are only guaranteed convergence to a bounded region [Gordon, 2001].

The experiments in this appendix parameterise $Q(\theta_t, i_t, u_t)$ using the lookup table scheme of Section 3.4.1. That section also defines how to compute $\nabla Q(\theta_t, i_t, u_t)$ for our experiments. In our POMDP setting the state inputs i_t to SARSA are replaced by *virtual state* inputs (g_{t+1}, y_t). This will typically invalidate the convergence guarantees until (g_{t+1}, y_t) is an unambiguous indicator of the state i_t. If $\mathcal{G}$ and $\mathcal{Y}$ are not finite, function approximation may be used at the risk of convergence failure. Empirical evidence [Loch and Singh, 1998] suggests that SARSA is fairly tolerant of partial observability.

C.1.2 The IGPOMDP-SARSA Algorithm

The IGPOMDP-SARSA algorithm is similar to the IOHMM-GPOMDP algorithm because we interleave SARSA(λ) and IState-GPOMDP phases until the magnitude of the estimated gradients drop below a threshold. An important difference is that IGPOMDP-SARSA performs gradient ascent on the FSC $\omega(h|\phi, g, y)$, whereas IOHMM-GPOMDP performs gradient ascent on the policy $\mu(u|\theta, \alpha, y)$.

Recall from Section 2.4 that SARSA(λ) learns a Q-function $Q(i_t, u)$, which is the discounted value of choosing action u in world state i_t and then acting optimally. The first phase executes one episode of SARSA(λ) where the world state i_t is replaced by the *virtual state*, defined as the I-state/observation tuple (g_{t+1}, y_t), which we assume — often incorrectly — is a sufficient indicator of world state to allow the optimal action to be chosen. The virtual state combines the observation y_t with the new I-state chosen from $\omega(\cdot|\phi, g_t, y_t)$. SARSA episodes are limited to T_{sarsa} steps.

Because the Q-function replaces $\mu(u|\theta, h, y)$, we denote the parameters of the Q-function as θ and write the discounted value of action u as

$$Q(\theta, g_{t+1}, y_t, u).$$

The best action is $\arg\max_u Q(\theta, g_{t+1}, y_t, u)$, which is followed with probability $1-\epsilon$. A random action is taken with probability ϵ, a so called ϵ-*greedy* policy [Mitchell, 1997].

The second phase runs IState-GPOMDP (see Algorithm 2), now with the fixed policy $\mu(u|\theta, h, y)$ generated by SARSA. The gradients of the ϕ parameters are estimated for T_{grad} steps before terminating and performing the line search to find a good step size as described in Appendix B. During the estimation the same ϵ-greedy policy used by SARSA is used to choose actions, however ϵ should be reduced between episodes.

IGPOMDP-SARSA is summarised by Algorithm 8. Line 10 deserves further explanation: SARSA is an acronym for State-Action-Reward-State-Action, which summarises the quantities that SARSA uses to update its internal Q-function. The first state-action pair refer to the previous state and action, and the second pair refer to the current state and action. The reward is that received after performing the first action. Line 10 passes the virtual-state version of these quantities to Algorithm 7.

Even if the assumption that (g_{t+1}, y_t) is a reasonable indicator of state is false to begin with, IState-GPOMDP learns an FSC that reveals relevant hidden state so that the assumption should become more valid as learning progresses. At the beginning the policy may perform no better than a random policy, but at the very least should not prevent IState-GPOMDP from witnessing the full range of state trajectories. The ϵ-greedy policy is important to both SARSA and helps IState-GPOMDPto experience all possible I-state trajectories.

IGPOMDP-SARSA has two additional benefits apart from quickly learning relevant policies: (1) the IState-GPOMDP phase has a reduced number of parameters to estimate, requiring less statistical evidence to generate good gradients, creating a variance reduction effect; (2) using SARSA(λ) to learn $\mu(u|\theta, h, y)$ potentially moves the agent out of the regions of zero gradient discussed in Chapter 7.

Algorithm 8 IGPOMDP-SARSA

```
1: Given:
     • Parameterised class of randomised FSCs $\{\omega(h|\phi, g, y) : \phi \in \mathbb{R}^{n_\phi}\}$.
     • An instance of SARSA($\lambda$) with $|\mathcal{G}||\mathcal{Y}|$ virtual input states, satisfying the requirements of Algorithm 7.
     • Arbitrary initial world-state $i_0$ and I-state $g_0$.
     • Observation sequence $\{y_0, y_1, \dots\}$ generated by the POMDP, I-state sequence $\{g_0, g_1, \dots\}$ generated stochastically according to $\omega(\cdot|\phi, g_t, y_t)$, and action sequence $\{u_0, u_1, \dots\}$ generated randomly using an $\epsilon$-greedy function of the value maximising action.
     • SARSA sample length $T_{\text{sarsa}}$; SARSA parameter $\lambda$; and discount $\Gamma$.
     • IState-GPOMDP gradient estimation length $T_{\text{grad}}$; discount $\beta \in [0, 1)$; and step size $\gamma > 0$.
2: while $\|\Delta_{T_{\text{grad}}}\| > \epsilon$ do
3:   $t = 0$
4:   while $t < T_{\text{sarsa}}$ do
5:     Observe $y_t$
6:     Choose $g_{t+1}$ from $\omega(\cdot|\phi, g_t, y_t)$
7:     $u_t = \arg\max_u Q(\theta, g_{t+1}, y_t, u)$
8:     With probability $\epsilon$ override $u_t$ randomly
9:     Execute one step of SARSA($\lambda$) with input $((g_t, y_{t-1}), u_{t-1}, r_t, (g_{t+1}, y_t), u_t)$
10:    Receive $r_{t+1}$
11:    $t \leftarrow t + 1$
12:  end while
13:  Set $z_0 = 0$, and $\Delta_0 = 0$ $(z_0, \Delta_0 \in \mathbb{R}^{n_\phi})$
14:  $t = 0$
15:  while $t < T_{\text{grad}}$ do
16:    Observe $y_t$ from the world
17:    Choose $g_{t+1}$ from $\omega(\cdot|\phi, g_t, y_t)$
18:    $u_t = \arg\max_u Q(\theta, g_{t+1}, y_t, u)$
19:    With probability $\epsilon$ override $u_t$ randomly
20:    $z_{t+1} = \beta z_t + \frac{\nabla\omega(g_{t+1}|\phi, g_t, y_t)}{\omega(g_{t+1}|\phi, g_t, y_t)}$
21:    $\Delta_{t+1} = \Delta_t + \frac{1}{t+1}[r(i_{t+1})z_{t+1} - \Delta_t]$
22:    $t \leftarrow t + 1$
23:  end while
24:  $\phi \leftarrow \phi + \gamma\Delta_{T_{\text{grad}}}$
25: end while
```

C.1.3 Convergence of IGPOMDP-SARSA

IGPOMDP-SARSA should always converge to a local maximum of the average reward if we meet the convergence requirements of SARSA(λ), which require that $\lambda = 0$ and that we use a lookup-table parameterisation for the Q-function. It does not suffer the problem exhibited by IOHMM-GPOMDP, where re-estimation of I-state model can cause a drop in η, because the I-state re-estimation must maximise the reward rather than its ability to just predict rewards. This guarantees that the next episode of SARSA(0) will begin with η at least as high as the end of the previous episode. Since lookup table SARSA(0) converges to the optimal policy, and IState-GPOMDP converges to a local maximum of η, we obtain overall convergence to a local maximum of the average reward. Convergence of lookup-table SARSA(λ) for $\lambda > 0$ is an open question [Sutton, 1999].

C.1.4 Preliminary Experiments

In this section we verify empirically that using SARSA(λ) to learn $\mu(u|\theta, h, y)$ simplifies the task of estimating gradients with respect to ϕ, observed as a reduction in the bias and variance of the estimates. An intuitive reason to expect better results is the reduced number of parameter gradients that are being computed, requiring less data to produce accurate estimates. The improved relevance of the sample trajectory due to the non-random policy should also reduce the bias and variance. We demonstrate this on our favourite toy scenario: Load/Unload (see Figure 2.2(a)).

C.1.4.1 Experimental Protocol

With $|\mathcal{G}| = 4$, as in Section 8.1, we compared the bias and variance of estimates of $\nabla^\phi \eta$ when gradients were estimated using IGPOMDP-SARSA, IState-GPOMDP, and IState-GPOMDP with a deterministic $\mu(u|\theta, h, y)$ set before hand. The latter experiment aims to verify that results are better when $\mu(u|\theta, h, y)$ is learnt by SARSA(λ) and that improved results are not merely due to fewer parameters being estimated.

As usual, $\omega(h|\phi, g, y)$ was parameterised by a lookup table of real numbers. We used exactly the same scheme, detailed in Section 3.4.1, to parameterise $Q(\theta, g_{t+1}, y_t, u)$. This fitted naturally into the SARSA(λ) algorithm which requires $\nabla^\theta Q(\theta, g_{t+1}, y_t, u)$.

In the first experiment we varied the discount factor β, keeping the number of gradient estimation steps fixed at $T_{\text{grad}} = 100,000$. In the second we increased the number of gradient estimation steps with fixed $\beta = 0.95$. The parameters ϕ and θ were initialised randomly to values in $[-0.5, 0.5]$. One episode of SARSA(λ) was completed before estimating the gradient. One difficulty with IGPOMDP-SARSA is determining

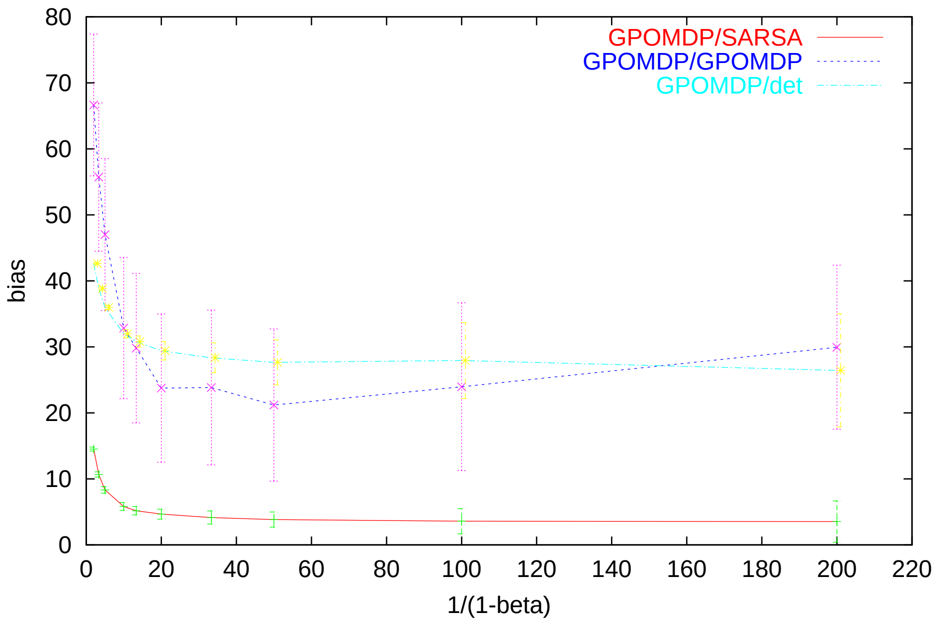


Figure C.1: Bias and variance of gradient estimates as β varies. The "GPOMDP/GPOMDP" curve uses IState-GPOMDP to estimate gradients w.r.t. θ and ϕ, the "GPOMDP/det" curve fixes θ and learns ϕ, the "GPOMDP/SARSA" curve uses SARSA(λ) to learn θ and IState-GPOMDP to learn ϕ. Results are averaged over 500 runs, error bars show 1 standard deviation. A value of 200 on the $\frac{\beta}{1-\beta}$ axis equates to $\beta = 0.995$.

sensible values for the SARSA(λ) parameters as well as the gradient estimation parameters. After some trial and error, we obtained the best results with $\Gamma = 0.95$, $\lambda = 0.8$ and $\epsilon = 0.2$. SARSA(λ) episodes were $T_{\text{sarsa}} = 100,000$ steps long.

C.1.4.2 Results

The results are shown in Figures C.1 and C.2. A significant reduction in both bias and variance can be seen in both graphs when using SARSA(λ) to learn $\mu(u|\theta, h, y)$. In particular, we see that we obtained a gradient estimate with less than 10° bias after only 500 steps when $\beta = 0.95$, but pure IState-GPOMDP had a bias of around 15° after over 200,000 steps.

C.1.5 Discussion

Despite promising early results we do not know how the benefit of this method will vary between different POMDPs. We expect it to be of greatest benefit at the start of learning. Once a reasonable policy is learnt the bias and variance of IState-GPOMDP

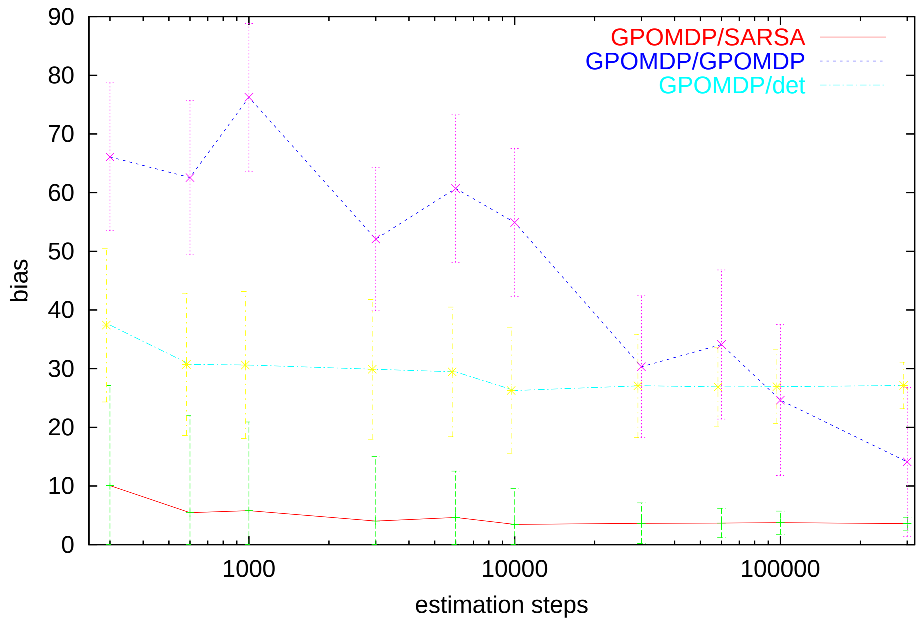


Figure C.2: Bias and variance of gradient estimates as the estimation time T_{grad} increases. Results are averaged over 500 runs.

tend to drop dramatically due the increased relevance of the sample trajectories. Much more investigation is needed to determine how useful this method is generally. In particular, because the SARSA and IState-GPOMDP phases gather independent sample trajectories, it is not clear if the total number of sample steps will be reduced by using IGPOMDP-SARSA.

IGPOMDP-SARSA is most likely to be useful for initialising $\mu(u|\theta, h, y)$ in preparation for running pure IState-GPOMDP. Using value methods to initialise policies before running policy-gradient algorithms was considered by Roy and Thrun [2001], although not in the context of agents with memory.

We could also use SARSA(λ) to learn the FSC, resulting in a pure value-function approach to learning FSC agents for solving POMDPs. This is similar to the approach of Peshkin et al. [1999] where one instance of SARSA is used to learn policies which can set external memory bits.

C.2 Over-Fitting on Fixed Random Sequences

We observed that policy-gradient methods using fixed random number generators can introduce false local maxima. In this section we describe how local maxima are in-

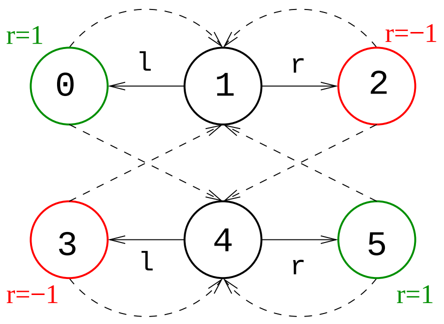


Figure C.3: A completely unobservable POMDP for which $\eta = 0$ for all policies. The actions are $\mathcal{U}$ ={`l, r`}, representing moving left or moving right from state 1 and 4.

troduced and demonstrate the over-fitting effect empirically. Using fixed random sequences seems too useful a method to ignore but we should be aware of the pitfalls.

Our example for this section is a simple 6 state *completely unobservable* MDP, shown in Figure C.3. In states 1 or 4 the agent controls which state it moves to, though when moving left it does not know if it will transit to state 0 or 3, and when moving right it does not if it will transit to state 2 or 5. All dashed transitions are followed with probability 0.5 regardless of the action.

All policies have an expected η of 0, however, when training with a fixed sequence of random numbers we observe that the algorithms in this thesis consistently converge to a small positive reward. The reward obtained is higher for shorter gradient estimation times. After entering states 0, 2, 3, or 5 there is always a random transition to state 1 or 4. If there are 100 estimation steps, then 50 random numbers are used to decide if state 1 or 4 should be entered. Suppose 26 times the choice is to enter state 1, and 24 times the choice is to enter state 4. If the policy is to always choose the move left action, then the agent will receive 26 rewards of +1 and 24 rewards of −1, so $\eta = (26 - 24)/100 = 0.02$. When the same sequence of random numbers is used for all gradient estimates, the agent learns that the random sequence results in state 1 being more likely to occur, and adjusts the policy toward always moving left. As soon as the learnt policy is tested on different sequences it will not consistently achieve $\eta > 0$.

This an instance of over-fitting. The agent is learning to take advantage of noise in the training sequences that does not reflect the true distribution of observation/action histories. We trade off variance reduction for a new form of bias.

Alternatively, if a free sequence of random numbers is used — meaning the random number generator is not reset between gradient estimates — then small biases in the random numbers are cancelled out. For example, if the next sequence of 100 random numbers results in 24 transitions to state 1 and 26 transitions to state 4, then the effect of the previous bias will be undone. The following sections present experimental evidence of the effect.

C.2.1 Experimental Protocol

We used memory-less IState-GPOMDP to learn a lookup table $\mu(u|\theta, y)$ with 2 parameters representing the probability of choosing to move left or right. No form of memory would allow a consistent positive reward to be achieved for truly uniform transitions to states 1 and 4. We performed 1000 independent runs, each with a different sequence of random numbers which is re-used for every gradient estimate during that run. The runs used an estimation time of $T = 100$. The results were not sensitive to β except for values close to 1; we used $\beta = 0.9$. No quadratic penalty was used.

C.2.2 Results

The results are summarised by the "Fixed RNG" curve in Figure C.4. It shows how η is a function of the amount of deviation from uniform exhibited by the random sequences. For example, the x-axis point $100|\pi_1 - \pi_4| = 10$ shows that $\eta \approx 0.05$ when averaged over all runs for which state 1 was entered 10 instances more or less than state 4. The number of runs falling into each point on the x-axis is not uniform. We have plotted the count of runs against $100|\pi_1 - \pi_4|$, shown by the "Samples" curve.

For comparison we performed 1000 independent runs using different random sequences for every gradient estimate during a run. The results are shown by the "Free RNG" curve. In this case the deviation from uniform exhibited by the random sequences changes during the run. Hence, the data point generated by each run is just the last estimate of η and whatever value of $100|\pi_1 - \pi_4|$ the random sequence for that estimate satisfied. It shows that the average η for all 1000 runs is approximately 0, a true indication of the agents' performance.

We repeated the experiment for an estimation time of $T = 1000$ steps. The results are shown in Figure C.5. It shows similar results except that $\eta = 0.021$ averaged across all 1000 runs, compared to $\eta = 0.063$ for $T = 100$, satisfying the expectation that longer sequences reduce over-fitting.

C.2.3 Discussion

It might be expected that all runs using a fixed random sequence should converge to the deterministic policy which chooses the action that gets the positive reward from the dominant state. If this were the case the "Fixed RNG" results in Figures C.4 and C.5 would be a straight line along $\eta = |\pi_1 - \pi_4|/T$. Some data points almost satisfy this constraint, for example, points in Figure C.5 for which $1000|\pi_1 - \pi_4| < 10$, and the

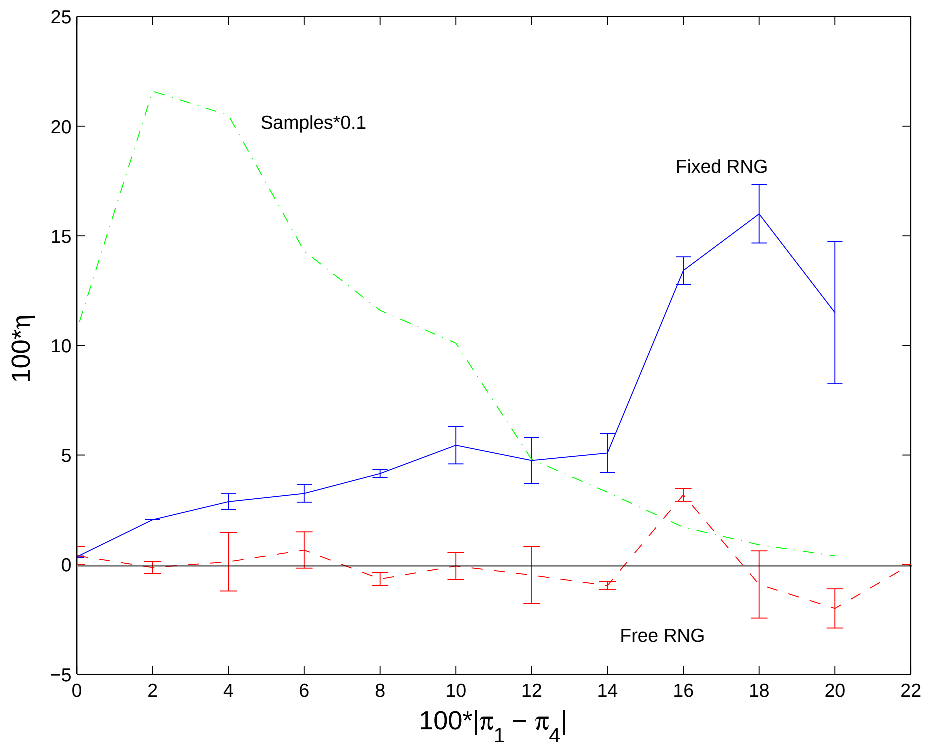


Figure C.4: The effect of using a fixed sequence of 100 random numbers on the POMDP in Figure C.3. The x axis measures the difference between the number of times states 1 and 4 were entered. The y axis is equivalent to the number of $+1$ rewards minus the number of -1 rewards. Results are based on 1000 runs of IState-GPOMDP. Error bars represent the standard error. The $-\cdot$ curve multiplied by 10 shows how many of the 1000 runs fell into each data point, that is, how may of the 1000 random sequences satisfied $x = 100|\pi_1 - \pi_4|$.

last point $1000|\pi_1 - \pi_4| = 78$. There are two reasons why other points do not follow this line. The first is simply the normal bias and noise in the gradient estimates. The second is a competing over-fitting effect. Suppose $\pi_1 > \pi_4$, then the policy should converge to "always move left." However, it may be that transitions into state π_4 are more often than not followed by large random numbers that promote generation of action `r`, providing a reward which promotes the policy "always move right."

The example, while easy to understand, has not demonstrated that over-fitting can cause poor agents to be learnt instead of optimal ones. However, we have already seen an instance of this: the Heaven/Hell experiments of Section 8.2. In this scenario the agent that fails to learn to visit the signpost cannot achieve an average reward of $\eta > 0$. Surprisingly, Table 8.2 shows that IState-GPOMDP obtains a mean η of 0.000178. Moreover, all IState-GPOMDP runs that failed to learn to visit the signpost learnt near deterministic policies that always move left, or always move right, at the

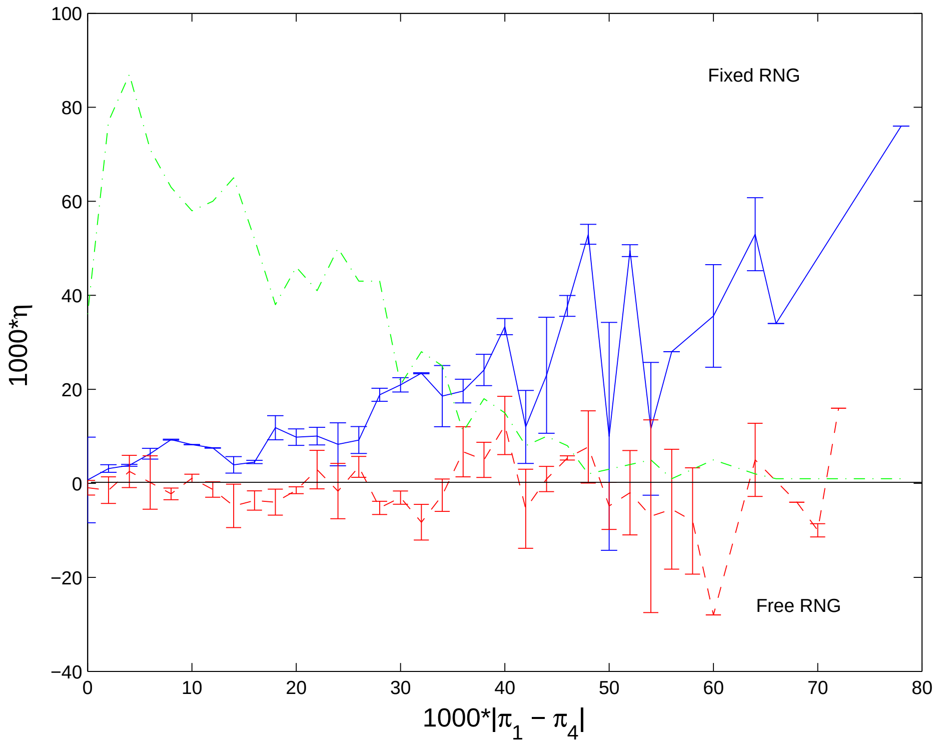


Figure C.5: The effect of using a fixed sequence of 1000 random numbers on the POMDP in Figure C.3. The $-\cdot$ curve shows how many of the 1000 runs fell into each data point, that is, how may of the 1000 random sequences satisfied $x = 1000|\pi_1 - \pi_4|$.

intersection. During early training the gradient prompting the agent to the correct policy can be dominated by the gradient pointing to the false local maximum. This occurred even for massive estimation times of $T = 10^7$. Without using fixed random sequences successive gradient estimates were sometimes more than 90° apart during early training, even for $T = 10^7$.

It was this quirk of the Heaven/Hell training that led to the investigation of the over-fitting phenomenon. Another problem from the literature which occasionally falls into this trap is the Tiger scenario [Cassandra, 1998], though only for very short estimation times of around $T = 100$.

We emphasise that it is the relative size of the true gradient to the effects of the finite sample size that dictates if over-fitting on the samples is likely to occur. This means that the over-fitting phenomenom is primarily due to small gradients which are problematic for all gradient-based optimisation algorithms. However, using fixed random number sequences can exacerbate the problem.

The discussion of this section applies equally to finite-horizon scenarios simulated using m sequences of fixed random numbers. It also applies to all Monte-Carlo methods for which the true state is not known. Even if the agent knows the belief state, two important states can be aliased while a fixed random sequence makes one slightly more likely to occur during simulation. One measure to prevent over-fitting is to periodically change the random number sequence used, for example, at the end of each line search phase of gradient ascent.

Appendix D

Hidden Markov Models

Hidden Markov models (HMMs) are a favoured model for time varying signals, notably speech. They are one solution to the problem of time-dependence in speech, which although theoretically flawed [Bourlard and Morgan, 1994], has proven empirically successful.

The notation for this appendix, in the context of the rest of the thesis, assumes that the HMM is modelling the unobservable world-state process. The HMM generates symbols that model the observation process. This is basically the approach of Chrisman [1992]. The HMM-states are labelled with $i \in \mathcal{S}$ and $j \in \mathcal{S}$ and the HMM outputs are labelled with $y \in \mathcal{Y}$. We retain the classical description of HMMs in terms of a state transition matrix A and a symbol emission matrix B. All symbols will be redefined so this appendix can be read independently of the rest of the thesis.

The following section defines Hidden Markov Models, followed by a basic description the Baum-Welch training procedure. Section D.1.2 describes how to segment a signal using trained HMMs and the Viterbi algorithm. We finish with a description of an HMM extension, called Input/Ouput HMMs, which can be driven by an input sequence. Most of the information and notation for this section comes from Rabiner and Juang [1986]. Other good references are Paul [1990] and Poritz [1988]. Appendix E can be viewed as a speech processing extension to this appendix. It discusses some more advanced HMM training methods and the drawbacks of using HMMs for speech. It also describes the close link between many ANN algorithms and HMM algorithms.

D.1 Standard Hidden Markov Models

A Markov model with $|\mathcal{S}|$ states $\mathcal{S} = \{1, \ldots, |\mathcal{S}|\}$ is defined by (A, π_0), where A is the stochastic transition matrix

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1|\mathcal{S}|} \\ \vdots & \ddots & \vdots \\ a_{|\mathcal{S}|1} & \cdots & a_{|\mathcal{S}||\mathcal{S}|} \end{bmatrix},$$

and the elements a_{ij} describe the probability of transition $i \to j$. The column vector $\pi_0 = [\pi(1), \ldots, \pi(|\mathcal{S}|)]'$ is the distribution of HMM-state occupation at $t = 0$. Ideally, this would be initialised to the initial belief state vector b_0. At each time step we make a transition from the current state i to the next state j, according to the probability distribution selected from row i of A.

A Hidden Markov Model has the same underlying structure as a Markov Model, except we cannot observe the state directly. There is a second stochastic process that emits a symbol every time we arrive in a state j.[1] We can observe the symbols y_t but not the true state i_t of the underlying Markov model.

Definition 5. *A discrete symbol HMM contains*

1. $|\mathcal{S}|$ *states* $\mathcal{S} = \{1, \ldots, |\mathcal{S}|\}$*;*
2. $|\mathcal{Y}|$ *symbols* $\mathcal{Y} = \{1, \ldots, |\mathcal{Y}|\}$ *that can be emitted;*
3. *an* $|\mathcal{S}| \times |\mathcal{S}|$ *stochastic transition matrix* A*;*
4. *an* $|\mathcal{S}| \times |\mathcal{Y}|$ *stochastic emission matrix* B*;*
5. *initial state distribution* π_0*.*

An HMM is described by $m = (A, B, \pi)$*.*

The new B parameter is the matrix that describes the probability of emitting symbol $y \in \mathcal{Y}$ when in state $j \in \mathcal{S}$

$$B = \begin{bmatrix} b_{11} & \cdots & b_{1|\mathcal{Y}|} \\ \vdots & \ddots & \vdots \\ b_{|\mathcal{S}|1} & \cdots & b_{|\mathcal{S}||\mathcal{Y}|} \end{bmatrix}.$$

HMMs are similar to POMDPs except that the HMM model is generative, that is, it is not driven by an input sequence such as the actions chosen by an agent. Figure D.1 shows a 3 state HMM with a_{ij} values labelled on the state transition arcs. The b_{jy} symbol distribution values are listed for each state and each of the 3 symbols. HMMs may also use continuous valued emissions, in which case B describes a continuous distribution for each state, usually in the form of means, variances and mixture weights for a mixture of Gaussians. As the model steps from state to state it emits a stream of symbols $\bar{y} = \{y_1, y_2, \cdots\}$. Practical symbol sequences for training are finite, ending at some fixed time limit, or when the model reaches some terminating state. Infinite horizon POMDPs are approximated by truncating the sequence after T steps.

[1]Conventions other than symbol emission upon entering a state are common.

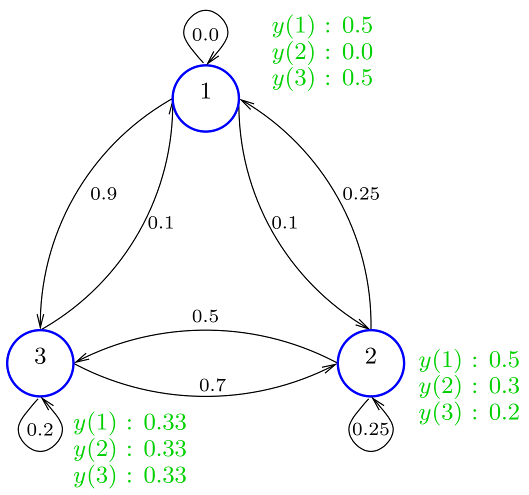


Figure D.1: Example 3 state hidden Markov model. The symbol emission probabilities in all states sum to 1. All the transitions out of a state also sum to 1.

D.1.1 The Baum-Welch Algorithm

Training an HMM involves maximising the likelihood that the HMM models the training sequence. The Baum-Welch method is popular for training HMMs. There are variations that make it gradient based or extend it to different emission probability densities [Poritz, 1988]. Some of these are discussed in Appendix E. The Baum-Welch method is an instance of Estimation-Maximisation (EM) training [Dempster et al., 1977]. It is an iterative algorithm, repeated multiple times over a training sequence $\bar{y}$. During model training, we are attempting to learn at the A and B matrices. For a model with $|\mathcal{S}|$ states and $|\mathcal{Y}|$ symbols, this represents $|\mathcal{S}|^2 + |\mathcal{S}||\mathcal{Y}|$ parameters.

Training starts by observing a symbol stream $\bar{y}$, generated by the underlying state transition process. We assume there is an underlying target HMM that stochastically generates symbols according to the true parameters m^*. We attempt to learn a model $m = m^*$. Standard Baum-Welch requires us to define m to have a fixed number of states $|\mathcal{S}|$ before hand, effectively assuming m^* has the same number of states. The number of symbols $|\mathcal{Y}|$ must also be defined, which is non-trivial if we are using $|\mathcal{Y}|$ discrete symbols to quantise a real-valued observation.

For each time step t in the sequence we estimate two quantities from the training data: $\alpha_t(i)$ and $\beta_t(i)$. The quantity $\alpha_t(i)$ is the probability of being in state i at time t, assuming we have moved forward through a series of t transitions. Because α describes the distribution of state occupation from the initial distribution π_0 forward to time t, it is known as the *forward* probability. It is recursively defined as

$$\begin{aligned} \alpha_0(i) &= \pi_0(i) \\ \alpha_{t+1}(j) &= \left(\sum_{i \in \mathcal{S}} \alpha_t(i) a_{ij} \right) b_{jy_t}. \end{aligned} \tag{D.1}$$

Strictly speaking, the forward probability and all the other quantities to be defined should include dependences on $\bar{y}$ and the current model m, in the same way that we included the dependence on $\bar{y}$ and ϕ in Equation (6.2). However, to simplify notation, we shall treat all quantities as vectors of values rather than conditional probabilities. The dependencies are always implied.

Define $\mathcal{S}_f \subseteq \mathcal{S}$ to be the states the system is allowed to terminate in. The quantity $\beta_t(i)$ is the probability of moving backwards from one of the states in $\mathcal{S}_f$ at time T to some other state i at time t. The value of β is defined recursively as

$$\begin{aligned} \beta_T(i) &= \begin{cases} 1 & , \ i \in \mathcal{S}_f \\ 0 & , \ \text{otherwise} \end{cases} \\ \beta_t(i) &= \sum_{j \in \mathcal{S}} \beta_{t+1}(j) a_{ij} b_{jy_t}. \end{aligned} \tag{D.2}$$

At the end of a sequence, the algorithm uses α and β to deterministically update A and B in a way guaranteed to converge to a local maximum. Several intermediate values are computed to perform the parameter updates, starting with the overall probability of being in state i at time t

$$\delta_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\Pr(\bar{y}|m)},$$

where the denominator is included as a normalising constant. Similarly, we define the probability of making transition $i \to j$ at time t as

$$\zeta_t(i,j) = \frac{\alpha_t(i) a_{ij} b_{jy_t} \beta_{t+1}(j)}{\Pr(\bar{y}|m)}. \tag{D.3}$$

To maximise $\Pr(\bar{y}|m)$ the parameter updates are derived from the definitions of the A and B matrices, where the values given above are used to compute the new model values. The a_{ij} update is

$$a_{ij} = \frac{\sum_{t=0}^{T-1} \zeta_t(i,j)}{\sum_{t=0}^{T-1} \delta_t(i)}, \tag{D.4}$$

and the b_{jy} update is

$$b_{jy} = \frac{\sum_{\{t:y_t=y\}} \delta_t(j)}{\sum_{t=0}^{T-1} \delta_t(j)}.$$

Emission probabilities are also commonly modeled by mixtures of Gaussians, with the advantage that continuous distributions can modeled. The update for the mean μ_j and variance υ_j of a single real number y_t modeled by a single Gaussian is

$$\mu_j = \frac{\sum_{t=0}^{T-1} \delta_t(j) y_t}{\sum_{t=0}^{T-1} \delta_t(j)} \tag{D.5}$$

$$\upsilon_j = \frac{\sum_{t=0}^{T-1} (y_t - \mu_j)^2}{\sum_{t=0}^{T-1} \delta_t(j)}. \tag{D.6}$$

The extension to multiple variables and mixtures of Gaussians is relatively straight forward and can be found in Poritz [1988]. Training can end in a number of ways, including checking for convergence of the parameters, or using a cross-validation. Baum-Welch maximises the probability $\Pr(\bar{y}|m)$, the probability that the symbol sequence $\bar{y}$ fits the model m. This represents a maximum likelihood training procedure rather than maximising the preferred posterior probability. The latter can be obtained using Bayes Rule provided the model priors are known. The value of $\Pr(\bar{y}|m)$ is given by

$$\Pr(\bar{y}|m) = \sum_{i \in \mathcal{S}_f} \alpha_T(i), \tag{D.7}$$

which is the total probability of being in any of the final states at the end of the symbol stream. The forward probability α is important for many sections of this thesis. In practice it is important to re-scale the forward probability at each time step to avoid numerical instability when the probabilities become very small.

The HMM training algorithm can be recovered from Algorithm 9 (the Baum-Welch training procedure for *Input/Output* HMMs) by setting $|\mathcal{U}| = 1$.

D.1.2 The Viterbi Algorithm

During classification we may not know the signal boundaries that delimit segments representing a particular model m_t. In practice all the trained models are concatenated together into a meta model. The most probable path through the meta model can be computed using the Viterbi algorithm [Viterbi, 1967]. The algorithm tells us which state is the most likely to be occupied at time t, hence which model we are most likely in at time t, hence segmenting the signal.

The Viterbi algorithm computes the likelihood of the *best* path through the states rather than likelihood of *all* paths through the states given by Equation (D.7). Therefore, it is an approximation to $\Pr(\bar{y}|m)$. Despite this, it can still be used to train HMMs. The Viterbi probability is the forward probability $\alpha_t(i)$ but the summation in

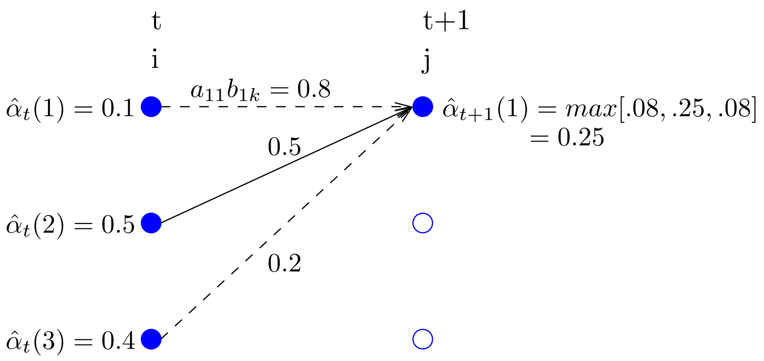

t
t+1
i
j
$\hat{\alpha}_t(1) = 0.1$
$a_{11}b_{1k} = 0.8$
$\hat{\alpha}_{t+1}(1) = max[.08, .25, .08]$
$= 0.25$
0.5
$\hat{\alpha}_t(2) = 0.5$
0.2
$\hat{\alpha}_t(3) = 0.4$

Figure D.2: We select which phoneme i was most likely to make a transition into phoneme j. The most likely state j defines the most likely trajectory back through the state lattice.

Equation (D.1) is replaced by a max operator

$$\begin{aligned}\hat{\alpha}_0(i) &= \pi_0(i)\\ \hat{\alpha}_{t+1}(j) &= \max_i\,[\hat{\alpha}_t(i)a_{ij}]b_{jy_t}.\end{aligned} \tag{D.8}$$

In words, $\hat{\alpha}_{t+1}(j)$ is computed by iterating through all possible next states j, and picking which current state i is most likely to make a transition into j, taking into account the probability that the HMM was in state i to start with. Figure D.2 illustrates this.

Every time the Viterbi decoder chooses a transition from state i_t to state i_{t+1} it stores the transition value of i_t, the predecessor node, in a table indexed by the time $t+1$ and the current node i_{t+1}. Once the end of the observation sequence has been reached we can evaluate the most likely final state $i_T^* = \arg\max_i \hat{\alpha}(i)$. The table then allows us traverse back through all the states that led up to state i_T^*, defining the most likely state sequence.

It is a feature of the Viterbi algorithm that it finds the most likely state sequence, which is different to the sequence of most likely states based on evaluating the full forward probability $\alpha_t(i)$ for each t.

Viterbi decoding can also be used as an error correction mechanism in signal processing. By limiting the length of the Viterbi lattice to l steps, we can find a path through a sequence of states by locking in states that fall off the end of the lattice. Although not optimal, it is a vast improvement over locking in the most likely state at the current time, equivalent to $l = 0$. More details of this last approach can be found in Section 10.3.1.2.

D.2 Input/Output HMMs

If we define an HMM with multiple transition matrices in a set $\mathcal{A}$, but still only one set of states, we can use an *input* signal to select transition matrices, driving the state transitions instead of letting them evolve purely stochastically. The HMM still emits a stream of *output* symbols.

Following our convention of the last section, where we attempt to learn a model that represents the evolution of world-states, we set the output symbols to be $y \in \mathcal{Y}$, and now the driving signals are the akin to the actions $u \in \mathcal{U}$. We do not consider how the input signal is generated and we assume that $\{y_1, \ldots, y_T\}$ and $\{u_1, \ldots, u_T\}$ are known prior to a training episode.

This system is illustrated in Figure D.3. If the HMM receives input $u_t = 0$ then we lookup the probability distribution for the next state from row i_t of transition matrix A_0. If $u_t = 1$ we use A_1, and so on. A single B emission matrix still generates symbols y depending on the current HMM state. This is a simple lookup-table instance of the more general *input/output* hidden Markov model (IOHMM) described by Bengio and Frasconi [1995, 1996]. This system also fits into the framework of *conditional random fields* developed by Lafferty et al. [2001] and McCallum et al. [2000]. In particular, the paper by McCallum suggests the possibility of using the Baum-Welch algorithm for training, just as we present in this section. We will prefer the term IOHMM to refer to an HMM that is driven by an input sequence.

Using the notation defined in Chapter 2 we could express the set of transition matrices $\mathcal{A}$ as $q(j|i, u)$ and the emission matrix B as $\nu(y|i)$. However, we continue with the matrix notation we used to describe HMMs.

This setting is different from the IOHMM classification experiments of Section 10.2.2 where we swapped the input and output sequences so that the observations y were the input, and the actions (classification labels) u were the outputs. In that case the IOHMM is acting as the POMDP agent rather than the partially observable world Markov process.

So far we have only re-cast IOHMMs as HMMs with multiple transition matrices. More formally:

Definition 6. *An input/output hidden Markov model (IOHMM) has:*

1. $|\mathcal{S}|$ *states* $\mathcal{S} = \{1, \ldots, |\mathcal{S}|\}$ *s.t.* $i_t \in \mathcal{S}$ *is the current state and* $j = i_{t+1}$ *is the next state;*

2. $|\mathcal{Y}|$ *symbols* $\mathcal{Y} = \{1, \ldots, |\mathcal{Y}|\}$ *encoding the output signal such that* $y_t \in \mathcal{Y}$ *is the output symbol at time* t*;*

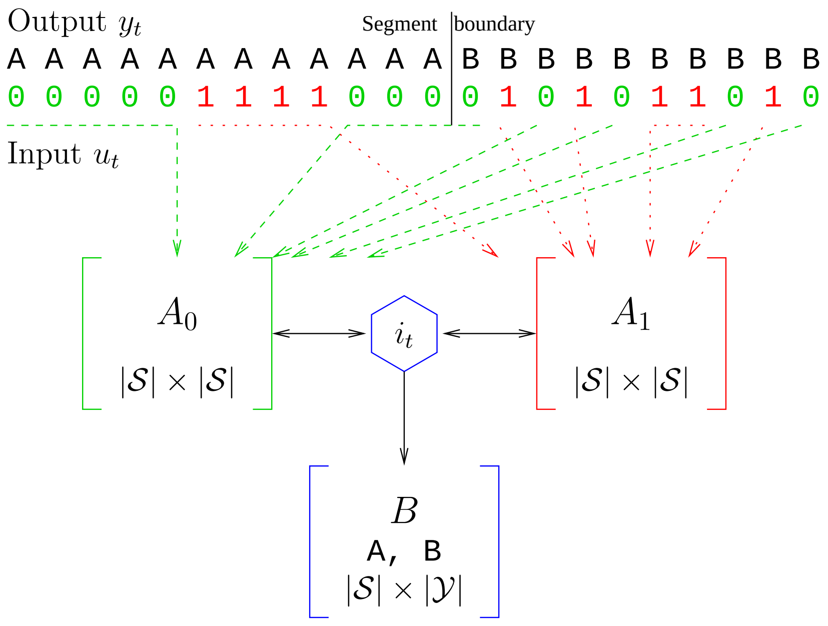

Output y_t
Segment
boundary
A A A A A A A A A A A A B B B B B B B B B B
0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 1 0 1 1 0 1 0
Input u_t
A_0
$|\mathcal{S}| \times |\mathcal{S}|$
i_t
A_1
$|\mathcal{S}| \times |\mathcal{S}|$
B
A, B
$|\mathcal{S}| \times |\mathcal{Y}|$

Figure D.3: Illustration of an IOHMM. The inputs switch between transition matrices and the IOHMM learns emission distributions for output observations.

3. *$|\mathcal{U}|$ symbols $\mathcal{U} = \{1, \ldots, |\mathcal{U}|\}$ encoding the input signal such that $u_t \in \mathcal{U}$ is the input at time t;*

4. *a set of $|\mathcal{Y}|$ independent $|\mathcal{S}| \times |\mathcal{S}|$ transition matrices $\mathcal{A} = \{A_1, \ldots, A_{|\mathcal{U}|}\}$;*

5. *a single $|\mathcal{S}| \times |\mathcal{Y}|$ stochastic emission matrix B;*

6. *initial state distribution π_0.*

A IOHMM is completely described by $m = (\mathcal{A}, B, \pi)$.

HMMs attempt to generate or predict an output signal given a model [Saul and Rahim, 1997], whereas IOHMMs are driven by the input signal while generating the output. IOHMMs can be seen as a discriminative version of HMMs. We can train a single large IOHMM to perform the same task as a number of normal HMMs. For example, an IOHMM can be trained to perform speech recognition by emitting speech labels based on the input wave-form. This is the setup used in Section 10.2.2 for the IOHMM classification of source HMMs experiment. Alternative modifications to HMMs attempt discriminative training by using a maximum mutual information criterion instead of ML [Reichl and Ruske, 1995, Warakagoda, 1996], however these systems still train an independent model per class, as discussed in Appendix E.

With minor changes to the Baum-Welch training algorithm we can use it to train IOHMMs. The first change is to the forward and backward probabilities defined by

Equation (D.1) which must now index the correct transition matrix using the input symbol u_t

$$\begin{aligned} \alpha_0(i) &= \pi_0(i) & \beta_T(i) &= 1 \\ \alpha_{t+1}(j) &= \left(\sum_{i \in \mathcal{S}} \alpha_t(i) a_{u_t ij} \right) b_{jy_t}, & \beta_t(i) &= \sum_{j \in \mathcal{S}} \beta_{t+1}(j) a_{u_t ij} b_{jy_t}. \end{aligned} \tag{D.9}$$

In this case the history $\bar{y}$ is assumed to include the history of actions as well as observations, that is, $\bar{y}_T = \{(y_1, u_1), \ldots, (y_T, u_T)\}$. The probability of emitting sequence $\{y_1, \ldots, y_T\}$ given input sequence $\{u_1, \ldots, u_T\}$ is

$$\Pr(\bar{y}|m) = \sum_{i \in \mathcal{S}} \alpha_T(i).$$

The arc probability given by Equation (D.3) changes to

$$\zeta_t(i, j) = \frac{\alpha_t(i) a_{u_t ij} b_{jy_t} \beta_t(j)}{\Pr(\bar{y}|m)}.$$

The calculation of $\delta_t(i)$ is unchanged. The updates to each a_{uij} must account for only the transitions $i \to j$ made using matrix A_u. This is achieved by summing over only the steps $\{t : u_t = \tilde{u}\}$ instead of all times as used in Equation (D.4), using the Markov property that these transitions are independent of any others

$$a_{\tilde{u}ij} = \frac{\sum_{\{t:u_t = \tilde{u}\}} \zeta_t(i, j)}{\sum_{\{t:u_t = \tilde{u}\}} \delta_t(i)}.$$

The b_{jy} update is unchanged. IOHMM training is summarised by Algorithm 9. The only modification needed for single Gaussian emission distributions is to replace line 20 with Equations (D.5) and (D.6).

Algorithm 9 Baum-Welch training of IOHMMs

```
1: Given:
     • Initialised transition matrices $\mathcal{A} = \{A_1, \dots, A_{|\mathcal{U}|}\}$.
     • Initialised emission matrix $B$.
     • Initial state occupancy probabilities $\pi_0(i)$.
     • Training sequence $\bar{y} = \{(y_1, u_1), \dots (y_T, u_T)\}$.
2: while error decreases on a cross validation test do
3:     $\alpha_0(i) = \pi_0(i)$
4:     for each state $j$ at each time step $t = 0$ to $t = T - 1$ do
5:         $\alpha_{t+1}(j) = \left(\sum_{i \in \mathcal{S}} \alpha_t(i) a_{u_t ij}\right) b_{jy_t}$
6:     end for
7:     $\beta_T(i) = 1$
8:     for each state $i$ at each time step $t = T - 1$ to $t = 0$ do
9:         $\beta_t(i) = \sum_{j \in \mathcal{S}} \beta_{t+1}(j) a_{u_t ij} b_{jy_t}$
10:    end for
11:    $\Pr(\bar{y}|m) = \sum_{i \in \mathcal{S}} \alpha_T(i)$
12:    for each state $i$, $j$ at each time step $t = 0$ to $t = T - 1$ do
13:        $\zeta_t(i,j) = \frac{\alpha_t(i) a_{u_t ij} b_{jy_t} \beta_t(j)}{\Pr(\bar{y}|m)}$
14:        $\delta_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\Pr(\bar{y}|m)}$
15:    end for
16:    for each $i$, $j$, $\tilde{u}$ do
17:        $a_{\tilde{u}ij} = \frac{\sum_{\{t:u_t=\tilde{u}\}} \zeta_t(i,j)}{\sum_{\{t:u_t=\tilde{u}\}} \delta_t(i)}$
18:    end for
19:    for each $j$, $\tilde{y}$ do
20:        $b_{j\tilde{y}} = \frac{\sum_{\{t:y_t=\tilde{y}\}} \delta_t(j)}{\sum_{t=0}^{T-1} \delta_t(j)}$
21:    end for
22: end while
```

Appendix E

Discriminative and Connectionist Speech Processing

This appendix provides background into existing speech methods that are related to those in Chapter 10, and more particularly, the large vocabulary continuous speech recognition experiments of Section 10.3. We focus on so called *discriminative* methods. They seek to rectify a failing of simple HMM implementations: each model is trained *independently* on the subset of the data that pertains to that model. Due to training errors and insufficient data this can result in failures arising from an incorrect model giving a higher likelihood than the correct model, even though the likelihoods for both models peak on the data that they model. Discriminative methods make models compete: while maximising the likelihood of the correct model they also minimise the likelihood of all other models on that data. This appendix can be read independently of the thesis body since it defines its own notation.

E.1 Overview

Discriminative methods for speech include using training criteria such as MMI (maximum mutual information) and MCE (minimum classification error) during the training of HMMs (hidden Markov models). These criteria will be explained in detail in Section E.3. Connectionist methods bring to mind the use of ANNs (artificial neural networks). HMMs and connectionist methods are closely related, sharing common solutions for tackling the complex problem of how to design MAP (maximum a posteriori) classifiers for speech. For example, the MMI training criterion can be applied to both ANN training and to discriminative HMM training. Also, ANNs can be trained to directly output the maximum a posteriori probability of each output class given the input vector [Mitchell, 1997], which is inherently discriminative. In addition, several authors have shown that it is possible to specify an ANN architecture equivalent to discriminative HMM training [Bridle, 1990, Niles and Silverman, 1990, Young, 1990].

Discriminative HMM methods begin to look, in theory, synonymous to connectionist approaches.

ANNs were studied intensively for speech processing in the late 1980s and early 1990s before losing popularity in the face of better empirical results from pure HMM approaches. At the this time there seems to be little interest in pure ANN approaches. However, there is interest in hybrid ANN/HMM approaches [Schuster, 1999, Haton, 1997, Bourlard and Morgan, 1998].

In this survey we briefly present basic approaches to discriminative training methods for HMMs and ANNs. We also compare these approaches, finding strong similarities between them. Then we look at methods that try to combine the best of both: hybrid HMM/ANN models trained with discriminative methods. The message of this survey is that traditional HMM approaches are flawed and that discriminative approaches, particularly those using hybrid approaches, may offer significant advantages.

Familiarity is assumed with the basics of both ANNs and HMMs. Appendix D is a brief introduction to HMMs. Many good papers can be founds on these topics, including Paul [1990], Poritz [1988], Rabiner and Juang [1986] for HMMs and Haykin [1999], Müller et al. [1995] for ANNs.

E.2 The Speech Problem

Speech recognition is the problem of choosing

$$m^* = \arg\max_m \Pr(m|O_u), \tag{E.1}$$

where $O_u = \{O_u(1), \ldots, O_u(T_u)\}$ is a time sequence of speech frames associated with utterance u and the m's are the models or classes that categorise the data. The correct model given the data is m^*. The model may represent a phone, word, speaker or some other such unit depending on the problem at hand. This is the MAP (maximum a posteriori) criterion for selecting the correct model. Using Bayes' rule we can transform (E.1) into

$$m^* = \arg\max_m \frac{\Pr(O_u|m)\Pr(m)}{\Pr(O_u)}. \tag{E.2}$$

Since all data is assumed equally likely $\Pr(O_u)$ is constant. If we further assume that all models (or classes) are equally likely, then we end up with the ML (maximum likelihood) criterion

$$m^* = \arg\max_m \Pr(O_u|m), \tag{E.3}$$

which can be interpreted as saying *assuming a model, what is the probability that the given data belongs to it?* By itself this is not a discriminative method since the models do not compete to classify the data. Instead, each model is trained individually to maximise the probability that it generates only the data for that model. The ML criterion is the one used in standard Baum-Welch training of HMMs [Rabiner, 1989]. Once the ML models have been trained we could use Equation (E.2) to compute the discriminative probability $\Pr(m|O_u)$. However, the individually trained models produce only estimates of $\Pr(O_u|m)$, trained on limited amounts of data, and only to a local maximum. Combined with errors in estimating $\Pr(m)$ — possibly from a different source than the spoken training data — it is preferable to perform training that directly estimates $\Pr(m|O_u)$, or at least trains the likelihood models to not only maximise $\Pr(O_u|m^*)$ but at the same time minimise $\Pr(O_u|m)\ \forall m \neq m^*$.

Since speech is a signal rather than a static pattern (E.3) should really be expressed as probabilities of sequences of observations and models

$$\begin{aligned}\mathbf{m}^* &= \arg\max_{\mathbf{m}} \Pr(O_u|\mathbf{m}) \\ &= \arg\max_{\mathbf{m}} \Pr(O_u(1), \dots, O_u(t)|m_1, m_2, \dots, m_t).\end{aligned}$$

This expression finds most probable sequence of models up to time t having seen all the data up to that time and assuming there is a model associated with each time step. This is the computation performed by HMMs — or more precisely, Viterbi decoding — where the model at each time step is a state of the HMM. HMMs are based on the assumption that the Markov property holds for speech, which can be phrased as *the most probable model (or state) depends only on the current observation and the previous model*

$$m_t^* = \arg\max_{m_t} \Pr(x_t, |m_t, m_{t-1}),$$

and the probability of the sequence is the sum over all possible model trajectories, where the probability of a trajectory is the product of the probability of each step given only the current model and the previous model

$$\Pr(O_u|\mathbf{m}) = \sum_{\forall m_1,\dots,m_{T_u}} \prod_{t=1}^{T_u} \Pr(x_t|m_t, m_{t-1}). \tag{E.4}$$

So now observations are assumed independent in time and the next model is assumed dependent only on the previous model. It is these assumptions that make HMM training tractable since (E.4) can be computed with a dynamic-programming approach [Bourlard and Morgan, 1998]. The simplifications of (E.3) and (E.4) admit good empirical results while allowing real-time processing. Unfortunately, the simplifications deliberately make untrue assumptions about speech [Rabiner, 1989]. This is not just true of HMMs since ANN approaches typically make similar assumptions [Bourlard and Morgan, 1994]. However, in ANNs we have the ability to relax these assumptions more readily than we do in HMMs. For example, to incorporate dependence on n previous models instead of just a single model, we can add $O(nM)$ inputs to the network. To accomplish this with an HMM with M models we need to create $O(M^n)$ individual models. This kind of increase in complexity is seen when HMMs move from modelling context independent phones (approximately 61 for the TIMIT corpus) to triphones where around 5000 models are used even after the unlikely or unhelpful triphones are removed [Lee, 1989, Hwang and Huang, 1993].

E.3 Discriminative Methods for ANN and HMM training

In this section we briefly describe two popular methods for performing discriminative training that can be applied to both ANNs and HMMs. We roughly follow the notation and structure of [Reichl and Ruske, 1995] which presents both methods in a consistent framework.

E.3.1 Maximum Mutual Information

The basic idea of MMI estimation is to maximise the extent to which knowing the data helps us to know which model is correct. An alternative view is to look at MMI as maximizing the ratio of the correct model likelihood to all other models, weighted by the class probabilities.

$$\frac{\Pr(O_u|m^*)}{\sum_{i=1}^{M}\Pr(m_i)\Pr(O_u|m_i)}.$$

MMI estimation methods are discussed and applied in too many papers to enumerate however some of the better descriptions are found in Reichl and Ruske [1995], Rabiner [1989], and Warakagoda [1996]. MMI methods can be applied to the language modelling phase of speech systems as well as the low level signal models [Ney, 1997].

In information theory mutual information is defined as

$$\begin{aligned} I(X,Y) &= H(X) - H(X|Y), \text{ where} \\ H(X) &= -\sum_{x\in X}\Pr(x)\log\Pr(x) \end{aligned} \tag{E.5}$$

which is the entropy of the discrete random variable X. Another interpretation of $H(X)$ is as the expected amount of information in X where the information carried by event $X = x$ is measured as $I(x) = -\log_b \Pr(x)$. If $b = 2$ the information is measured in bits. $I(X,Y)$ tells us to what extent knowing X helps us to know Y. To apply this to speech with the intent of training model m^*, let $Y = m^*$, and take the information expectation over the data in the set $X = \{O_u : m^* = \arg\max_m \Pr(m|O_u)\}$, that is, all the training data available for model m^*. Maximising (E.5) increases how much X tells us whether the model is m^*.

$$\begin{aligned}
I(X,m^*) =& H(X) - H(X|m^*) \\
=& -\sum_{O_u \in X} \Pr(O_u) \log \Pr(O_u) + \sum_{O_u \in X} \Pr(O_u) \log \Pr(O_u|m^*) \\
=& \sum_{O_u \in X} \Pr(O_u) \left[\log \Pr(O_u|m^*) - \log \sum_{i=1}^{M} \Pr(m_i) \Pr(O_u|m_i) \right] \\
=& \sum_{O_u \in X} \Pr(O_u) \log \left(\frac{\Pr(O_u|m^*)}{\sum_{i=1}^{M} \Pr(m_i) \Pr(O_u|m_i)} \right). \qquad \text{(E.6)}
\end{aligned}$$

Where the last summation can be interpreted as a sum over the data for each time step. The second line makes use of the fact that

$$\Pr(O_u) = \sum_{i=1}^{M} \Pr(O_u, m_i) = \sum_{i=1}^{M} \Pr(O_u|m_i) \Pr(m_i).$$

The speech technology community generally assumes that all observations are equally probable and defines I with respect to a single observation O_u, in which case (E.6) simplifies to

$$I(O_u, m^*) = \log \left(\frac{\Pr(O_u|m^*)}{\sum_{i=1}^{M} \Pr(m_i) \Pr(O_u|m_i)} \right), \qquad \text{(E.7)}$$

and from this form comes the intuition that MMI maximises the ratio of the correct model likelihood to the likelihood of *all* models. Also, $I(O_u, m^*) + \log(\Pr(m^*))$ gives us $\Pr(m^*|O_u)$, the more desirable MAP criterion. We can also relate the MMI criterion to the idea of minimising cross entropy. Equation (E.6) can be re-written as

$$-I(O_u, m^*) = \sum_{O_u \in X} \Pr(O_u) \log \left(\frac{\Pr(O_u)}{\Pr(O_u|m^*)} \right),$$

which is the calculation for discrete cross entropy [Rabiner, 1989]. Thus, maximising mutual information can be re-cast as minimising cross entropy, which can be thought of as minimising the difference between the distribution of the data, and the data given the model [Bridle, 1992].

E.3.1.1 Gradient Descent for MMI

Suppose we have some parameterised approximator (or possibly an approximator for each model) that computes $\Pr(O_u|m, \theta)$, where θ represent the parameters of the system. By computing the gradient of $-I(O_u, m)$ with respect to θ we can train our approximator to perform speech processing according the MMI criterion. To reduce clutter we label the numerator of (E.7) as L_*, the likelihood of the correct model, and L_a as the combined likelihood of all models, giving

$$L_* = \Pr(O_u|m^*), \qquad L_a = \sum_{i=1}^{M} \Pr(m_i) \Pr(O_u|m_i).$$

Rewriting (E.7) to be suitable for minimisation we have

$$\begin{aligned} -I(O_u, m^*) &= \log \frac{L_a}{L_*} \\ I(O_u, m^*) &= \log L_a - \log L_* \\ \frac{-\partial I(O_u, m^*)}{\partial \theta} &= \frac{1}{L_a}\frac{\partial L_a}{\partial \theta} - \frac{1}{L_*}\frac{\partial L_*}{\partial \theta}. \end{aligned}$$

Provided we can compute $\partial L_*/\partial\theta$ and $\partial L_a/\partial\theta$ gradient descent can be used to optimise parameters θ.

How do we apply this to a real system? One approach is to use knowledge of the model priors $\Pr(m_i)$ (or assume they are uniform), and instead of approximating $\Pr(O_u|m_i)$ we approximate the posterior probability $\Pr(m_i|O_u)$ using one large ANN where each output represents a model. Interpreting ANN outputs as probabilities is explained in Section E.4.1. This approach is used in Alphanets [Bridle and Dodd, 1991, Bridle, 1992, Niles and Silverman, 1990] and in several RNNs (Recurrent Neural Networks) [Robinson, 1994, Wei and Vuuren, 1998, Fallside, 1992]. Examples of approximating $\Pr(O_u|m)$ with ANNs are rare since we might expect a single network to share information more efficiently than m independent networks, requiring fewer parameters and consequently requiring less training data. One example (that uses the MCE described below rather than MMI) is Lee et al. [1995], described further in Section E.4.3.1.

The difficulty with speech for ANNs is the time varying nature of the signal. The question is how to represent O_u to the ANN so that it outputs a sequence of model probabilities. ANNs usually assume a static pattern, however speech consists of a possibly continuous stream of data broken down into frames of around 10 ms, each with tens to hundreds of features [Lee, 1988]. A key difference between the various connectionist and hybrid inspired approaches is how they deal with the time varying nature of speech. The natural way HMMs handle time varying signals has contributed to their popularity.

E.3.1.2 MMI for HMMs

In general it is possible to use gradient ascent for training HMMs though care must be taken to maintain stochastic constraints. For example, the transition probabilities out of a state must sum to one. This can be achieved by mapping parameters in $\mathbb{R}$ to probabilities [Niles and Silverman, 1990, Bridle, 1990]. Huo and Chan [1993] point out that this method may introduce extra local maxima, which is undesirable since gradient methods only guarantee convergence to one of these local maxima. Alternatively, Lagrange multipliers can be used to perform gradient ascent subject to stochastic constraints [Rabiner, 1989, Huo and Chan, 1993].

The gradients of the discriminative cost functions described here can be incorporated into an HMM update gradient, or HMM training can be run as normal and then gradient descent on the discriminative objective function can be performed as corrective training [Normandin, 1991, Huo and Chan, 1993]. Alternatively Reichl and Ruske [1995], Normandin and Cardin [1992], and Gopalakrishnan et al. [1989] discuss methods that extend the Baum-Welch updates to rational objective functions, which are applicable to the objective functions outlined here.

All of these methods require the derivative of the cost function with respect to the HMM parameters. The following equations [Warakagoda, 1996] give the gradient of the MMI criterion with respect to the HMM parameters for the state transitions $i \to j$ of model m, denoted a_{ij}^m and the discrete observation probabilities for symbol $O_u(t)$ in state j, denoted $b_{jO_u(t)}^m$

$$\begin{aligned} -\frac{\partial I(O_u, m^*)}{\partial a_{ij}^m} &= \left(\frac{1}{L_a} - \frac{\chi_m(m^*)}{L_*}\right) \sum_{t=1}^{T} \alpha_{t-1}(i) b_{jO_u(t)}^m \beta_t(j), \\ -\frac{\partial I(O_u, m^*)}{\partial b_{jO_u(t)}^m} &= \left(\frac{1}{L_a} - \frac{\chi_m(m^*)}{L_*}\right) \frac{\alpha_t(j)\beta_t(j)}{b_{jO_u(t)}^m}, \end{aligned} \tag{E.8}$$

where $\alpha_t(i)$ is the forward HMM probability of being in state i at time t and $\beta_t(i)$ is the corresponding backwards probability. The identity function is represented by $\chi_x(y)$. The sum of these gradients across all the training data will result in the gradient of the negative of (E.6). The extension of (E.8) to the case of continuous densities represented by a single Gaussian is given by Bridle and Dodd [1991].

E.3.2 Minimum Classification Error

Minimum Classification Error seeks to minimise exactly what we care about, the empirical error rate. It is introduced and described in by Juang and Katagiri [1992] which also compares this criterion to standard error measures such as the mean squared error. A similar measure called Minimum Empirical Error was introduced by Ljolje et al. [1990]. The idea is to construct a distance measure between the probability of the correct choice and the probability of all other choices

$$d_*(O_u) = \Pr(O_u|m^*) - \left[\frac{1}{M-1}\sum_{m_i \neq m^*} \Pr(O_u|m_i)^\eta\right]^{\frac{1}{\eta}}. \qquad \text{(E.9)}$$

The parameter η can be thought of as adjusting the distance metric used. If $\eta = 1$ we have an L_1 norm and we are simply summing the probabilities of incorrect models. As $\eta \to \infty$ only the largest incorrect probability has any effect. We then use $-d_*(O_u)$ in a sigmoid function to construct a smooth cost function l that can be minimised in order to maximise Equation (E.9)

$$l(d_*(O_u)) = \frac{1}{1 + \exp(\gamma d_*(O_u))}. \qquad \text{(E.10)}$$

By summing Equation (E.10) over all the training data for each model we achieve an empirical estimate of the probability of misclassification. It is interesting to compare the MCE to MMI. If we set $\eta = 1$ and take the log of both terms in Equation (E.9) then we have

$$d_*(O_u) = \log\left(\frac{\Pr(O_u|m^*)}{\sum_{m_i \neq m^*} \frac{1}{M-1}\Pr(O_u|m_i)}\right).$$

Which differs from Equation (E.7) only in whether the correct model m^* is included in the summation and the assumption of uniform priors $\Pr(m_i)$. MCE is also very similar to the idea of distance normalisation discussed in [Furui, 1994].

E.3.2.1 Gradient Descent for MCE

In practice it seems more common to use the log form of Equation (E.9) [Lee et al., 1995, Reichl and Ruske, 1995], which results in the following gradient for $l(d_*(O_u))$

with respect to an arbitrary set of parameters θ

$$\frac{\partial l(d_*(O_u))}{\partial \theta} = \sum_{i=1}^{M} l(d_*(O_u))(1 - l(d_*(O_u)))G_i(O_u) \frac{\partial \Pr(O_u|m_i, \theta)}{\partial \theta} \tag{E.11}$$

$$G_i(O_u) = \begin{cases} \frac{-1}{\Pr(O_u|m_i)} & \text{if } m_i = m^* \\ \frac{\Pr(O_u|m_i,\theta)}{\sum_{j=1}^{M} \Pr(O_u|m_j,\theta)} & \text{otherwise,} \end{cases}$$

Summing Equation (E.11) over all the data for each model results in a gradient that minimises the probability of misclassification.

We are now subject to the same questions about how to approximate $\Pr(O_u|m, \theta)$ as we were in Section E.3.1.1, and we can apply the same solutions. Juang and Katagiri [1992] take the approach of training a single large ANN to approximate all the probabilities (see Section E.4.1). Lee et al. [1995] train a single ANN for each $\Pr(O_u|m, \theta)$ (see Section E.4.3.1). Using ANNs is convenient because we know how to compute the gradient of an ANN output with respect to its parameters.

E.3.2.2 MCE for HMMs

Reichl and Ruske [1995], and Nogueiras-Rodriíguez et al. [1998] use a gradient descent version of MCE estimation. Denoting the state of HMM m occupied at time t as q_t^m, the gradient for the state-specific observation densities is

$$\frac{\partial l(d_*(O_u))}{\partial \Pr(O_u|j, m)} = l(d_*(O_u))(1 - l(d_*(O_u)))G_i(O_u) \sum_{t:q_t^m = j} \frac{1}{\Pr(O_u(t)|j, m)}, \tag{E.12}$$

which sums up the gradient contributions for all observations associated with state j. In the case of discrete symbols we have $\Pr(O_u(t)|j, m) = b^m_{jO_u(t)}$. Equation (E.12) requires that the optimal state sequence is known, which can be determined using the Viterbi algorithm.

E.3.3 Results Comparison

Where possible this section provides comparative experimental results for the methods described so far. Most of the results use the TIMIT database [Garofolo et al., 1993]. However, due to factors such as varying definitions of accuracy and the varying levels of problem difficulty, the results should not be compared across different paragraphs. Unless otherwise stated, the results should be assumed to be defined as 100% - % deletions - % substitutions - % insertions, at the phoneme sequence level; although it is not always clear from the cited papers if this is the metric used.

A German speech database was used by Reichl and Ruske [1995] to compare MMI and MCE training of HMMs. Using the standard maximum likelihood criterion they achieved a 59.5% accuracy. Applying MMI improved this to 62.0% and MCE achieved 64.8%. They also reported that MMI training was less stable than MCE, requiring smaller step sizes.

On the TIMIT database with 39 phones Fallside [1992] demonstrated an RNN (recurrent neural network) system trained with MMI with a frame by frame accuracy of 75.1%. This is compared with the CMU Sphinx [Seymore et al., 1998] HMM system that achieved 73.8%.

On a Cantonese digit test set MCE improved results from the ML baseline of 82.9% to 90.0% [Lee et al., 1995]. This system used a small RNN for each digit. The same system applied to English digits resulted in recognition improving from 92.3% to 93.5%. The disparity in improvement arises from the inherent confusability of Cantonese digits that allows discriminative approaches to work well.

E.4 Neural Network Speech Processing

Why should we bother with ANNs if HMMs provide a good way to represent speech signals?

- ANNs can model arbitrary non-linear transformations of input parameters, including the ability to model arbitrary probability distributions [Bourlard and Morgan, 1998]. The most flexible pure HMMs typically assume probability distributions made up of mixtures of Gaussians with a covariance matrix that is 0 except along the diagonal.
- If trained properly, ANNs can directly estimate the discriminative MAP $\Pr(m|O_u)$ criterion (see Section E.4.1).
- ANN systems can be 2–5 times faster during recognition than traditional methods for equivalent performance [Schuster, 1998].
- A single ANN can be trained to do the same job as multiple HMMs, decreasing the overall number of parameters to be trained, and improving the use of training data [Schuster, 1998].
- ANNs can relax the Markov assumption by considering multiple frames of data (past and future) at once [Lippmann, 1989, Bourlard and Morgan, 1994]. It is difficult to do this with HMMs since it is necessary to minimise the dimensionality of observations and the number of states to allow estimation of the parameters with minimal data.

- ANNs can consider categorical inputs. For example, encoding psycho-acoustic features [Pols, 1997].

- ANNs can model arbitrary state durations, unlike HMMs in which durations follow an exponential model. This is important for normalising the likelihood contributions from short consonants against long vowels and other speech warping phenomenon. HMMs can be modified to model arbitrary durations, however the computational expense seems to outweigh the benefits [Rabiner, 1989]. Durations can also be modeled in a post-processing phase, but these methods appear ad-hoc, requiring extra weighting terms to be optimised.

- In practice, it appears necessary to carefully initialise the observation densities used in HMMs to achieve good performance [Rabiner, 1989, Bourlard and Morgan, 1998]. This is not the case with ANNs.

The disadvantages of ANNs include:

- The lack of a principled way to convert a sequence of observations into an optimal sequence speech units. There is a need to include some form of search for the globally optimal sequence of units given the local estimates of matches from an ANN. This is the function usually achieved by the Viterbi search in HMMs. Hybrid methods are a way to avoid this problem.

- ANN systems using gradient methods are 10–20 times slower to train than HMMs using Baum-Welch training since they are restricted to small steps in parameter space [Schuster, 1998]. Conjugate gradient [Fine, 1999] and line searches can speed up gradient ascent training. Approximate gradient ascent algorithms such as RPROP can also be used [Schuster, 1998].

- Speech ANNs have roughly 10 thousand to 2 million parameters [Schuster, 1998], requiring a large amount of data and cross-validation to avoid over-fitting.

- Some sources quote that state-of-the-art HMMs with sophisticated tied and interpolated distributions and thousands of context dependent models, have roughly 25% better error rate than the best ANN/Hybrid systems when sufficient training data is available [Bourlard and Morgan, 1998].

The rest of this section looks at different ways to contrive ANNs capable of handling speech data.

E.4.1 Big, Dumb, Neural Networks

Ignoring for the moment the problem of time dependence in speech, it is possible to view an ANN as performing a series of static probability estimation tasks. The input to the network is a frame of speech plus future and past frames of speech to provide context. Each output gives an estimate to the probability of a particular model given the input.

Consider ANN outputs $y_1, \ldots, y_M \in \mathbb{R}$; how do we interpret these outputs as probabilities? More specifically, how would we construct a network to compute the posterior probabilities $\Pr(m_i|O_u)$? A standard method for doing this is to use a soft-max distribution (3.8) at the output [Robinson, 1994, Morgan, 1994]. Assume that the network is learning to estimate the MAP probability $\Pr(m_i|O_u)$, then for each possible model $m_1, \ldots m_M$ we define

$$\Pr(m_i|O_u) := \frac{\exp(y_i)}{\sum_{j=1}^{M} \exp(y_j)}.$$

Given an arbitrary cost function J, such as $-I(X, m^*)$, we compute

$$\frac{\partial J}{\partial \theta_j} = \frac{\partial J}{\partial \Pr(m_i|O_u)} \sum_{k=1}^{M} \frac{\partial \Pr(m_i|O_u)}{\partial y_k} \frac{\partial y_k}{\partial \theta_j}. \tag{E.13}$$

The gradient of the soft-max distribution with respect to the network outputs is

$$\frac{\partial \Pr(m_i|O_u)}{\partial y_k} = \Pr(m_i|O_u)(\chi_i(k) - \Pr(m_k|O_u)) \tag{E.14}$$

which can be interpreted as driving the difference between the desired probability $\chi_i(k)$ and the actual output probability to zero. Once the gradient of the cost function with respect to the outputs is known it is straight forward to use back propagation to compute $\frac{\partial y_k}{\partial \theta}$. For example, consider the following simple cost function for the observation $O_u(t)$

$$J = -log \Pr(m_t^*|O_u(t), \theta) \tag{E.15}$$
$$\frac{\partial J}{\partial \Pr(m_t^*|O_u(t), \theta)} = -\frac{1}{\Pr(m_t^*|O_u(t), \theta)}.$$

This is the equation for the normalised-likelihood cost function [Richard and Lippmann, 1991]. It simply measures the log probability of utterance O_u assuming we know (or can estimate) the correct model m_t^*. Minimising this quantity will maximise the posterior probability of the correct model given the observation. Minimising the sum of (E.15) over time will maximise the log likelihood of the correct sequence of models.

Substituting Equations (E.15) and (E.14) into Equation (E.13), we obtain

$$\frac{\partial J}{\partial \theta_j} = -\sum_{k=1}^{M} (\chi_{m_t^*}(m_k) - \Pr(m_k|O_u(t), \theta)) \frac{\partial y_k}{\partial \theta}. \tag{E.16}$$

In the simple case where y_k represents the kth output of an ANN with linear output nodes and w_{hk} is the weight from hidden node h to output k, (E.16) simplifies to

$$\frac{\partial J}{\partial w_{hk}} = -(\chi_{m_t^*}(m_k) - \Pr(m_k|O_u(t), \theta))w_{hk}.$$

E.4.1.1 Alternative Cost Functions for MAP Estimation

Provided there is sufficient training data and the ANN is sufficiently complex to represent $\Pr(m_i|O_u(t))$, minimising Equation (E.15) will result in an ANN that estimates $\Pr(m_i|O_u(t))$. This is proved in [Richard and Lippmann, 1991], which also proves that the same is true of the mean square error cost function and the cross entropy function

$$\begin{aligned} J &= -\sum_{i=1}^{M} d_i \log \Pr(m_i|O_u(t), \theta) + (1 - d_i) \log(1 - \Pr(m_i|O_u(t), \theta)) \\ \frac{\partial J}{\partial \theta_j} &= -\sum_{i=1}^{M} \frac{d_i - \Pr(m_i|O_u(t), \theta)}{\Pr(m_i|O_u(t))(1 - \Pr(m_i|O_u(t)))} \frac{\partial \Pr(m_i|O_u(t))}{\theta_j}, \end{aligned}$$

where $d_i = 1$ if $m_i = m_t^*$ and 0 otherwise. Cross entropy has been popular for speech recognition applications, for example [Wei and Vuuren, 1998, Fallside, 1992].

The experimental comparisons of Richard and Lippmann [1991] indicate that these three cost functions produce similar results if enough training data is available. The cross entropy cost and normalised likelihood weight cost converged faster than MSE, and the normalised likelihood resulted in marginally better estimation accuracy in regions of low probability.

These cost functions differ from those covered in Section E.3 because they estimate posterior probabilities of models rather than likelihoods $\Pr(O_u(t)|m)$. Despite this, these training methods can be used to estimate scaled likelihoods simply by dividing the outputs by the prior probability of the models. This is discussed further in Section E.5.1.

E.4.1.2 Implementation Issues

In the case where unlabelled data is available, it is sufficient to train the system using as much labelled data as is available, then use the resultant classifier to label the unlabelled data, using this new larger labelled set to train a new classifier. This process is repeated, with each classifier bootstrapping off the labelling of the previous classifier, until no improvement is gained [Bourlard and Morgan, 1998].

In practice, networks that classify the 61 phone TIMIT database have several hundred inputs (including frames for context), 500–4000 hidden units, and 61 outputs, requiring in the order of 10^6 parameters [Morgan, 1994]. Training such networks provides interesting challenges [Aberdeen et al., 2000]. Once such a network has been trained some form of search is needed to compute the most likely temporal phone sequence from the individual frame probabilities. Alternatively, ANNs that handle time-series data can be used as discussed in the following sections.

E.4.2 Time-Delay Neural Networks

Time-delay neural networks (TDNNs) were one of the earliest attempts to modify ANNs to cope with sequences of inputs. The output of each node of a TDNN is the same as a standard ANN, the weighted sum of its inputs, but integrated over T_l frames. Each layer l may integrate over a different time period. Thus each node in layer l must have a local shift register to store the weighted sum of the last T_l inputs. If the input to a node at the current time t is $\mathbf{x}_t$, then the output is

$$y_t = \sum_{s=0}^{T_l-1} c_s f(\mathbf{x}_{t-s}),$$

where c_s is an optional weighting term for each past frame and f is the ANN activation function for that layer. Figure E.1 illustrates this idea. TDNNs can be thought of as integrating evidence for or against a class over a finite period of time. They are trained using a modified form of error back propagation. Good results were obtained for classifying plosive consonants using TDNNs compared to standard ANNs [Waibel, 1989]. They have also been used to approximately determine phone labels to use as discrete HMM symbols by Ma and Compernolle [1990]. This system recognised Dutch digits, discriminating between 21 phonemes. Results improved from 90% to 93% over a HMM with 200 discrete symbols from a codebook. A drawback of TDNNs is the fixed amount of memory for each node. This is somewhat rectified in [Lin, 1994] where TDNNs are extended to automatically adapt the value of T_l. TDNNs are further reviewed by Lippmann [1989], Haton [1997], and Bridle [1992].

E.4.3 Recurrent Neural Networks

RNNs avoid the main problem of TDNNs by allowing all previous inputs to effect the current output. Any traditional network architecture can be classed as an RNN if it involves feedback from the output back into the inputs and/or hidden units [Haton,

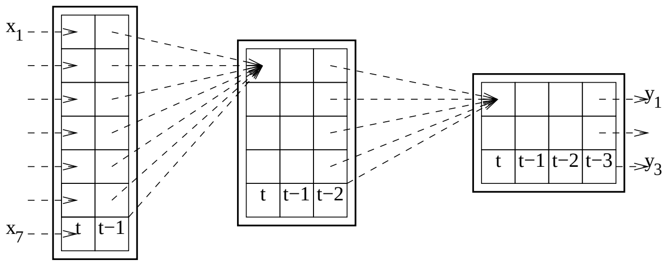


Figure E.1: A Time-Delay Neural Network with 7 inputs and 2 frames of memory, 5 hidden nodes with 3 frames of memory and 3 outputs with 4 frames of memory.

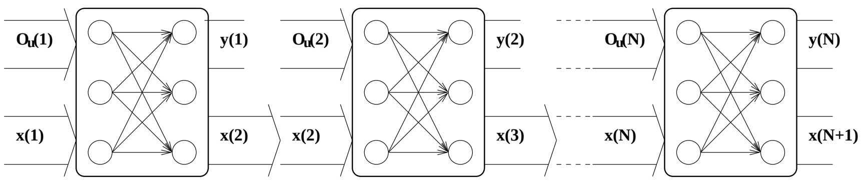


Figure E.2: An RNN unfolded N times for training.

1997]. In a typical application the inputs are augmented with a real-valued state vector which was output by the network at the previous time step. This implies that the outputs are augmented to provide the next state given the current input $O_u(t)$ and previous state $\mathbf{x}_t$. RNNs draw theoretical justification from their similarity to the feedback methods used in linear state-based control systems [Fallside, 1992].

The most common training method is *Back-Propagation Through Time* (BPTT), which unfolds the network N times and propagates the errors at the outputs $\mathbf{y}_t$ and state $\mathbf{x}_t$ back through each time step (see Figure E.2). The initial state can be set to an arbitrary fixed value. The state error at time N is zero because the final state has no effect on the classification being made. Unfortunately, this method limits the amount of past context the network can be trained to consider up to N frames. More complex methods exist which avoid this limitation [Robinson, 1994, Schmidhuber, 1992].

In Robinson [1994] an RNN was used to classify the 61 phone TIMIT database. The network was trained using the cross-entropy criterion described in Section E.4.1. This allowed the 61 outputs to be interpreted as $\Pr(m_i|O_u(1), O_u(2), \ldots, O_u(t))$ and fed into a Viterbi state decoder. Thus, this application is actually a hybrid approach. The state dimension was 176, with 47,400 parameters, trained with Back-Propagation Through Time using the fast RPROP error descent algorithm [Schuster, 1998]. Results show that 72.8% of phones were correctly identified by the RNN compared to 74.4% for a mono-phone HMM system trained with the MMI criterion. At the time the best results were from an HMM based system [Woodland, 2001] with 76.7%. RNNs are further reviewed by Lippmann [1989], Haton [1997], and Schuster [1999].

E.4.3.1 Modelling Likelihoods with RNNs

If we wish to approximate the likelihoods $\Pr(O_u|m_i)$ we could use a separate small ANN for each model and have a single output giving the probability of O_u. This is the approach used by Lee et al. [1995], where one RNN is trained for each of 11 Cantonese digits. The number of outputs is approximately the number of identifiable acoustic units, or states, that exist in the digit. At each time step the current state in each network is $q_j = \arg\max_i y_i$. A network has a high probability of being the correct model for the observations if the state index j increases monotonically to the final state along the duration of the utterance.

E.4.3.2 Bi-Directional Neural Networks

As they have been formulated above RNNs cannot take into account future frames of data in computing $\Pr(m|O_u)$. It is natural to expect that we need future context as well as past context to optimally identify a unit of speech. A simple way to provide future context is to train the network to delay its decision on frame t until frame $t + c$ where c is the number of future frames to consider. An alternative is to extend RNNs to allow all frames, past and present, to be considered. This architecture is called the the Bi-Directional RNN [Schuster, 1996, 1998, 1999]. BRNNs have two sets of state vectors, one for the forward time direction and one for the reverse time direction. At time t separate hidden layers compute the next forward and backward state vectors, while the output layer estimates $\Pr(m_i|O_u(1), O_u(2), \ldots, O_u(T_u))$ based on $O_u(t)$, the forward state computed at $t - 1$ and the backward state computed at $t + 1$. For real-time recognition some window of speech data needs to be considered if utterances are longer than a few tens of frames.

A subset of the TIMIT 61 phone database was trained using BRNNs in [Schuster, 1996] and compared to RNNs using delayed decisions of up to 4 frames. The best RNN actually had 0 delay with an accuracy of 51.2% using 8 state variables and 1518 parameters. The BRNN had an accuracy of 55.3% using 8 state variables in each direction and a total of 2182 parameters. The poor performance of the delayed RNNs in this experiment is not consistent with other results and may be due to the use of a small data training set and the extra difficulty in training for delayed decisions.

E.4.4 Representing HMMs as ANNs

Young [1990] emphasised the similarity between HMMs and RNNs. We do not usually attempt to interpret the value of an RNN state vector $\mathbf{x}_t$, but we might conceive of a

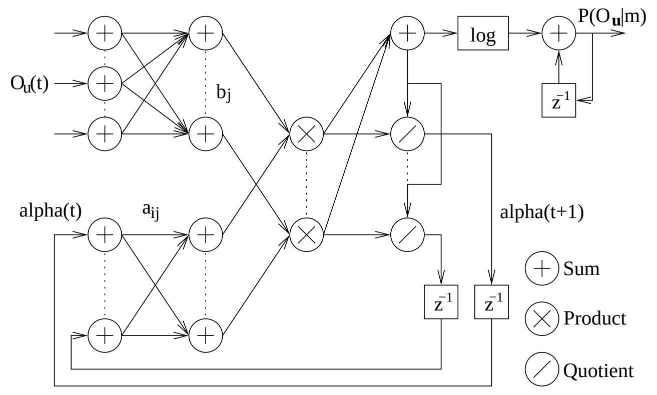


Figure E.3: An RNN implementation of an HMM.

network trained to produce a state vector equivalent to the forward state probability used in the Baum-Welch training procedure for HMMs, that is, $\mathbf{x}_t = \alpha_t$. The key idea is to note that the recursive forward probability computation for a single HMMs can be written as

$$\begin{aligned} \alpha_t(j) &= \sum_i \alpha_{t-1}(i) a_{ij} b_{jO_u(t)} \\ &= \sum_i \alpha_{t-1}(i) w_{ij}(t), \end{aligned} \tag{E.17}$$

which looks like the computation performed to compute the activation of an ANN node except that the weights are dependent on the current observation.

E.4.5 Alphanets

The concepts of the previous section are extended by Bridle [1990], and Bridle and Dodd [1991], resulting in Alphanets, and the work of Niles and Silverman [1990]. Both reach the conclusion that HMMs can be cast exactly as an RNN if we allow multiplication and division units as well as the standard summation units. The first two factors in Equation (E.17) can be computed with a feed forward pass where the inputs are α_t and the weights are the stochastic matrix elements a_{ij}. The observation probabilities $b_{jO(t)}$ can be estimated with another feed forward network where the inputs are $O_u(t)$ and there is an output for the probability given each state. The outputs are multiplied for each state, and then normalised to give α_{t+1}. The output of the network is the log sum of α_{t+1}. A possible network is shown in Figure E.3.

Applications based on the Alphanets methodology appear to assume one HMM state per acoustic model, using one network to represent one large HMM. They use discriminative forms of HMM training to update the parameters such as those discussed in Section E.3.1.2. Alphanets were not a new method for speech processing, rather, the importance of Alphanets is in providing a new view of existing HMM methods and unifying the idea of discriminative and connectionist approaches to speech [Bridle, 1992].

E.4.6 Other uses of Neural Networks for Speech

In this section we have focused on ANN methods that attempt to determine the probability of a model (such as a phone) given the observations O_u. However, ANNs can be used in speech processing in many other ways:

- *Phone recognition* – Alternative methods of performing phone recognition include the use of Kanerva models, Classification & Regression Trees and other ANNs with novel processing units [Fallside, 1992].
- *Vector Quantisation* – Learning vector quantisers based on self-organizing feature maps and other ANN approaches can be used to process observations to generate symbols for discrete HMMs [Kurimo, 1997, Anderson, 1994].
- *Pre-processing* – ANNs can perform arbitrary non-linear transformations of the input. This can perform tasks such as removing noise, or adapting to a new speaker [Huang et al., 1991, Ran and Millar, 1993, Zhang and Millar, 1994].
- *Hierarchical Mixtures of Experts* – Various expert classifiers including those discussed already can be combined through the use of a hierarchy of gating networks [Schuster, 1998, Ran and Millar, 1991] trained with the EM algorithm [Dempster et al., 1977].
- *Predictive Networks* – ANNs can be used to predict extra features. For example, they can be trained as auto-regressive models given previous observations and the current state [Bourlard and Morgan, 1998].
- *Language Modelling* – ANNs can be used to estimate the probabilities of sequences of phones, used for re-scoring N-best lists of phone sequences [Bourlard and Morgan, 1998].

E.5 Hybrid Speech Processing

In this section we present two of the most common hybrid approaches. The first is a broad approach, allowing many theoretically justified variations, and is the subject of active research. The second describes a method for globally optimising both the ANN and HMM components of hybrid systems.

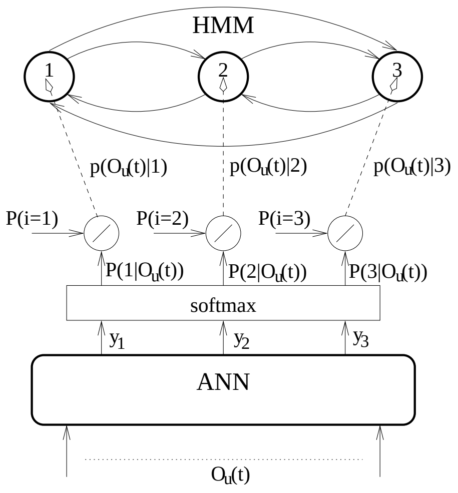


Figure E.4: Using an ANN to generate HMM observation likelihoods.

E.5.1 Estimating Observation Densities with ANNs

Each state i of an HMM is associated with a probability distribution over the observations $\Pr(O_u(t)|i)$. Section E.4 noted that an advantage of ANNs over HMMs is their ability to model an arbitrary distribution, non-linear in the inputs. A large body of work including Bourlard and Morgan [1998], Bourlard et al. [1995], Bourlard and Morgan [1994], and Morgan [1994] is devoted to this idea. Essentially, the methods of Section E.4.1 are applied to estimate observation likelihoods $p(O_u(t)|i)$, and those likelihoods are used in the HMM procedures in place of $b_{iO_u(t)}$ for equations such as (E.17).

However, the ANN methods of Section E.4.1 estimate the posterior $\Pr(m|O_u(t))$ rather than the likelihoods used by the HMM search. It is easy to convert from a posterior to a *scaled likelihood* by assuming $\Pr(O_u(t))$ is constant and using Bayes' rule

$$p(O_u(t)|m) \propto \frac{\Pr(m|O_u(t))}{\Pr(m)}.$$

This simply amounts to dividing the network output probabilities by the model priors estimated from the training data. This quantity can be used in Equation (E.17) as the model likelihood. In this context a model m and an HMM state i are synonymous, since the ANN estimates a phone probability and the HMM consists of one phone per state. Figure E.4 illustrates the process. One phone per state limits the applicability of the algorithm though in principle there is no reason why ANNs could not be used to estimate probabilities for multi-state phones. Once the ANN has been trained, an HMM training procedure can be used to estimate the state transition probabilities a_{ij}. On an 152 speaker subset of the TIMIT database with 63 phones, the method described above achieved a frame by frame accuracy of 54.8% with 351 inputs and 1024 hidden nodes [Bourlard and Morgan, 1994]. Each frame had 39 inputs and ± 4 frames of context were provided for a total of 351 ANN inputs. Single Gaussian per phone density estimates achieved 43.3%.

E.5.1.1 MAP Estimation with ANN/HMM Hybrids

It may seem counter-intuitive to estimate posterior probabilities, just to turn them into less informative scaled likelihoods. Let us return to the idea of using ANNs to estimate *state* probabilities for HMMs rather than just *emission* probabilities. By estimating state *transition* probabilities $\Pr(i_t|i_{t-1}, O_u(t))$ instead of just occupancy probabilities $\Pr(i_t|O_u(t))$ we can avoid computing the scaled likelihood. The probability of the model is the product of $\Pr(j|i, O_u(t))$ for each state i_t in the sequence, summed across all possible state sequences. Taking into account that HMM m may model extra language information not modeled by the ANN transition probabilities, we have [Bourlard and Morgan, 1998]

$$\Pr(m|O_u, \theta) = \Pr(m) \sum_{\forall i_1, \ldots, i_T} \left[\prod_{t=1}^{T} \Pr(i_t|i_{t-1}, O_u(t), \theta) \frac{\Pr(i_t|i_{t-1}, m)}{\Pr(i_t|i_{t-1})} \right]$$

and the Viterbi approximation replaces the sum with a maximisation at each step. $\Pr(i_t|i_{t-1}, m)$ is the state transition probability according to the model. $\Pr(i_t|i_{t-1})$ is the state transition probability estimated from the training data by counting transitions. The dependency on θ emphasises the dependency on the ANN parameters. If the HMM model simply models the transitions in the data we have $\Pr(i_t|i_{t-1}, m) = \Pr(i_t|i_{t-1})$ and the last factor vanishes. Otherwise, $\Pr(i_t|i_{t-1}, m)$ allows us to encode useful knowledge that may not be evident in the training data, for example, the task may involve a restricted set of words, altering the distribution of phones.

E.5.2 Global Optimisation of ANN/HMM Hybrids

An alternative ANN/HMM approach taken by Bengio et al. [1992]. This work views the ANN as mapping a high dimensionality set of frame data into a small set of continuous observations to be input to an HMM. The HMM estimates observation probabilities using a mixture of Gaussians. Used in this way the ANNs are performing regression rather than classification. Multiple networks can be used to pre-process the data in different ways, each concentrating on various hard to distinguish features. For example, one network is trained to produce observations particularly useful for difficult plosive classification, while another network may produce broadly useful features.

The clever aspect of this structure is that the gradients of the HMM parameters, computed in Section E.3.1.2, can be propagated back into the ANNs, simultaneously maximising the discriminative powers of the HMMs and the ANNs. This requires computing $\partial b_{jO_u(t)}/\partial y_k$, the derivative of the observation probability at time t for state j with respect to the kth output of the combined networks. Having derived this quantity back-propagation can be used to derive the gradients of the network weights. The same globally trained ANNs are used to provide observations for all states of all HMMs.

This method was evaluated using the TIMIT database with 7214 triphone models. Observation features were calculated with 3 networks: a recurrent network for broad features using 12 inputs, a recurrent network for plosive features using 74 inputs, and a linear network to combine the results of the first two. The result was 8 continuous observations for the HMMs. The networks use a total of 23,578 weights. The first two networks were pre-trained to perform recognition tasks. Each HMM had 14 states and 3 distributions, tied to the transitions between the states. Global optimisation boosted accuracy from 81% to 86%. A hybrid based on the ideas in Section E.5.1 achieved 74%.

E.6 Summary

This survey has described the popular MMI and MCE criteria for discriminative speech processing systems. We showed how either criteria can be applied to HMMs, ANNs, or some arbitrary parameterised probability estimator. Connectionist approaches to probability estimation were reviewed, including static multi-layer perceptrons, TDNNs, RNNs and Bi-Directional RNNs. The close link between HMM approaches and connectionist approaches was also explored by showing how ANN architectures such as Alphanets perform the same calculation as the HMM forward probability calculation. The advantages of ANNs for speech processing compared to HMMs were also listed, noting that while ANNs have many theoretical advantages over HMMs, taking advantage of them is difficult, requiring the training of very large ANNs. Finally, we described the hybrid methods that use ANNs to estimate phone likelihoods for HMM states, and methods that allow the global optimisation of HMM/ANN hybrids.

Appendix F

Speech Experiments Data

This appendix provides extra experimental details and results to go with Chapter 10.

F.1 Cases for HMM classification

The matrices below define the 4 sets of HMMs used in the discrimination tests of Section 10.2.2.1. All sets define 2 or 3 models, all with 2 states and 2 symbols. Common matrices are only shown once. The matrices that are different for models within the same test are differentiated with the letter subscript `A` for the first model, `B` for the second model, and so on. Test I (see Figure 10.2):

$$A_{\mathtt{A}} = \begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix} A_{\mathtt{B}} = \begin{bmatrix} 0.1 & 0.9 \\ 0.9 & 0.1 \end{bmatrix} B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Test II:

$$A_{\mathtt{A}} = \begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix} A_{\mathtt{B}} = \begin{bmatrix} 0.1 & 0.9 \\ 0.9 & 0.1 \end{bmatrix} B = \begin{bmatrix} 0.75 & 0.25 \\ 0.25 & 0.75 \end{bmatrix}.$$

Test III:

$$A_{\mathtt{A}} = \begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix} A_{\mathtt{B}} = \begin{bmatrix} 0.1 & 0.9 \\ 0.9 & 0.1 \end{bmatrix} A_{\mathtt{C}} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix} B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Test IV:

$$A_{\mathtt{A}} = \begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix} B_{\mathtt{A}} = \begin{bmatrix} 0.5 & 0.5 \\ 0.9 & 0.1 \end{bmatrix} A_{\mathtt{B}} = \begin{bmatrix} 0.1 & 0.9 \\ 0.9 & 0.1 \end{bmatrix} B_{\mathtt{B}} = \begin{bmatrix} 0.1 & 0.9 \\ 0.9 & 0.1 \end{bmatrix}.$$

F.2 Bakis Model Signal Segmentation

This experiment uses a similar procedure to the HMM discrimination experiments of Section 10.2.2.1. Hand crafted HMMs are used to generate segments of observations y_t. IState-GPOMDP is used to train an FSC that labels each observation with the HMM that emitted it. Following on from those experiments, we label the HMMs used in this experiment as Test V.

Test V was created to measure the ability of the system to split a continuous stream of observations into segments belonging to each model, that is, we do not assume the length of the segments is known. The HMMs for Test 5 are more complex than the previous tests. We use two Bakis type HMMs with 5 states and 5 discrete output symbols:

$$A = \begin{bmatrix} 0.9 & 0.1 & 0 & 0 & 0 \\ 0 & 0.9 & 0.1 & 0 & 0 \\ 0 & 0 & 0.9 & 0.1 & 0 \\ 0 & 0 & 0 & 0.9 & 0.1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$B^{\mathtt{A}} = \begin{bmatrix} 0.0 & 0.2 & 0.4 & 0.4 & 0.0 \\ 0.0 & 0.0 & 0.2 & 0.4 & 0.4 \\ 0.2 & 0.4 & 0.4 & 0.0 & 0.0 \\ 0.4 & 0.4 & 0.0 & 0.0 & 0.2 \\ 0.4 & 0.0 & 0.0 & 0.2 & 0.4 \end{bmatrix} B^{\mathtt{B}} = \begin{bmatrix} 0.0 & 0.2 & 0.4 & 0.4 & 0.0 \\ 0.0 & 0.0 & 0.2 & 0.4 & 0.4 \\ 0.4 & 0.0 & 0.0 & 0.2 & 0.4 \\ 0.4 & 0.4 & 0.0 & 0.0 & 0.2 \\ 0.2 & 0.4 & 0.4 & 0.0 & 0.0 \end{bmatrix}.$$

The Bakis HMM structure only allows self transitions and forward transitions. This structure has previously been used to model speech [Bakis, 1976]. The only difference between the two source HMMs is the emission distributions for states 3 and 5, making this a difficult problem.

F.2.1 Experimental Protocol

This experiment used an ANN with a single hidden layer of $n_h = 8$ hidden units. The hidden layer was squashed using the tanh function. The gradient time was $T = 1 \times 10^7$

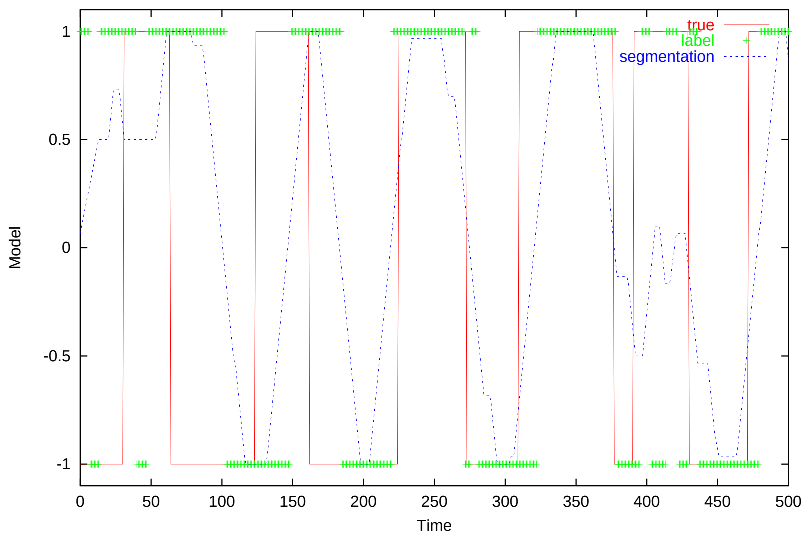


Figure F.1: Solid lines indicate which model is generating the observations, showing the correct label. Crosses indicate the label (action) output by the system at each time step. The dashed line indicates the rectangular window smoothed decision from Equation (10.2).

steps and the line search estimation time was 2.5×10^6 steps. The discount factor was $\beta = 0.9$. The quadratic penalty was $\wp = 0.0005$. The best number of I-states was found experimentally to be 16. A deterministic $\mu(u|g)$ was used as described in Section 10.2.1.1. The reward function was also unchanged with a reward of 1 for a correct label and -1 for an incorrect label.

F.2.2 Results

Results are shown in Figure F.1 for a fragment of the output of the best out of 10 trained agents. The crosses indicate the raw decision output by the system for each time step. The solid line indicates the true segment boundaries, and the dashed line is the output of the rectangular filter. The correct raw decision was made 77% of the time.

We experimented with using a $w = 30$ step rectangular window described by Equation (10.2). The window length was chosen from trials of $w = 10, 20, 30, 40$. The distance of the dotted line from 0 on the model axis can be interpreted as a confidence in the decision. Declaring a segment when the filtered decision crosses a threshold of ± 0.25 on the model axis correctly identified 10 segments with 2 deletions.

Table F.1: The mapping from the 48 numbered phonemes used in our experiments to the 63 TIMIT phonemes. The third column counts the instances of each phoneme in the training set.

Num.	*TIMIT phones*	*instances*	*Num.*	*TIMIT phones*	*instances*
0	b	1517	24	y	700
1	d	1672	25	hh, hv	1165
2	g	796	26	iy	3233
3	p	1790	27	ih	2837
4	t	2768	28	eh	2271
5	k	2666	29	ey	1599
6	dx	1344	30	ae	1622
7	q	1691	31	aa	1592
8	jh	710	32	aw	496
9	ch	573	33	ay	1355
10	s	4336	34	ah	1548
11	sh	943	35	ao	1343
12	z	2546	36	oy	215
13	zh	115	37	ow	1173
14	f	1562	38	uh	334
15	th	515	39	uw, ux	1385
16	v	1398	40	er, axr	2897
17	dh	1684	41	ax	2600
18	m, em	2470	42	ix	5073
19	n, en, nx	5287	43	pau, #h, h#	5851
20	ng, eng	857	44	epi	614
21	l, el	3736	45	pcl, tcl, kcl, qcl	8814
22	r	3278	46	bcl, dcl, gcl	4983
23	w	1556	47	ax-h	301

F.3 TIMIT Phoneme Mapping

The TIMIT continuous speech database [Garofolo et al., 1993] is labelled with 63 different phonemes. Some of these phonemes are allophones, or groups of phonemes that sound the same and rely on context to define them. For the low level phoneme recognition part of an LVCSR system it is usual to combine groups of allophones into one phoneme, eliminating the ambiguity that they represent. We used the mapping outlined in Lee [1989], given by Table F.1.

Appendix G

Bunyip Communication

This appendix discusses the communication costs associated with the distributed ANN training described in Section 11.5. We show that the communication costs for distributed training of an ULSNN are not trivial. A reduce algorithm optimised for Bunyip's topology is also described and some experimental timings given.

G.1 Communication Costs

The inter-process communication costs during ANN training arise from *broadcasting* the ANN parameters to all processes and *reducing* the ANN error and gradients from each process to the master process. The parameter broadcast is cheap, since many copies of the same data is sent to all processes. Broadcasts can take advantage of features such as TCP/IP broadcasting. The reduce process is more difficult with each process generating unique vectors that must be collected and summed by the master process. The time taken to reduce data grows with both the number of parameters and the number of processes. The remaining communication consists of start and stop messages which are negligible.

A typical neural network with 100 inputs, 50 hidden layer neurons, and 50 output neurons, requires 7500 parameters, or 30 KBytes of data (single precision), to be sent from every node to the master node after the gradient estimation has completed. A naive reduction over 194 processes using a 1 Gb/s link, such as used in Bunyip, would take 0.05 seconds assuming 100% network utilisation. Our ULSNN with 400 inputs, 480 hidden layer neurons and 3203 output neurons requires 1,729,440 parameters or 6.6 MBytes of data per process, which would require 10.1 seconds. There is sufficient memory on each node to occupy both processors for 446 seconds calculating gradients before a reduce operation is required. Consequently, the reduce operation would cost at least 2.3% of the available processing time, more if not enough training data is available or the network size is increased.

This demonstrates that although communication costs for distributed ANN training are minimal for commonly implemented network sizes, ULSNN training must optimise inter-process communication to achieve the best performance.

We reduced communication as much as possible by only distributing the neural-network parameters to all the slaves at the very start of training (rather than at each step), and thereafter communicating only the search direction and the amount to step in that direction. One significant reduce operation is required per epoch to send the error gradient vector from each process to the master. The master then co-ordinates the step size search with the slaves.

All inter-node communication was done using the LAM implementation of MPI [LAM Team, 1999]. Communicating parameters or directions to all processors required a 6.6 MBytes broadcast operation from the server to each of the 194 processors in the cluster, while reducing the gradient back to the master required 6.6 MBytes of data to be communicated from each process back to the server. LAM/MPI contains a library reduce operation that uses a simple $O(\log n)$ algorithm that distributes the load of the reduce over many processes instead of naively sending 194 gradient vectors to one node. This results in a reduce operation on Bunyip that takes 8.5 seconds over 8 stages.

G.2 Optimising Reductions

There are two problems with existing free implementations of MPI reduce operations. The first is the lack of shared memory protocols on clusters with multi-processor nodes. Instead, they use slow TCP/IP communication between processors on the same motherboard. Secondly, the reduce operation does not take advantage of the topology of the cluster. For example, the best reduce algorithm to use on a ring network might be to send a single vector to each node on the ring in turn, which adds its contribution before passing the vector to the next node. On a star network the best algorithm might be to send each contribution to the central server and sum as they arrive.

To decrease the time taken per reduce, a customised reduce was written, using shared memory for intra-node communication, and MPI non-blocking calls for inter-node communication. This routine is summarised by Figure G.1. It is split into 4 stages, each of which takes advantage of an aspect of Bunyip's topology shown in Figure 11.1.

1. Each node contains two processors, both running an instance of the training process. All 97 nodes (including the server), reduce 6.6 MBytes of data between processes by using shared memory, taking 0.18 seconds. The time taken to add the two sets of data together is approximately 0.005 seconds.

2. Each node in group A can open a 100 Mb/s connection to any node in group

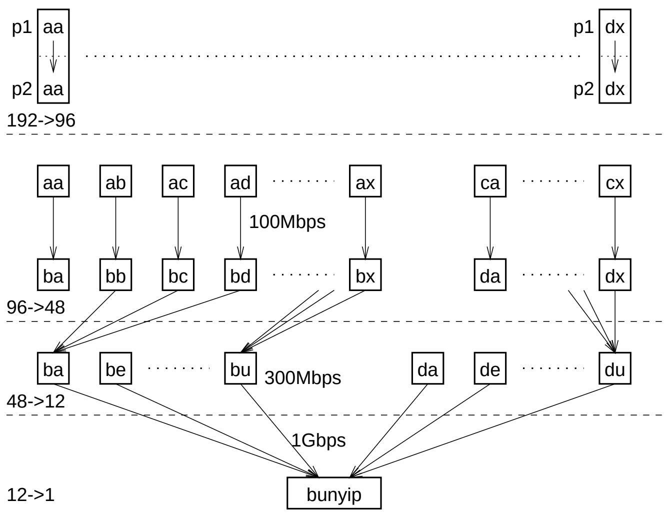


Figure G.1: The four stages of our customised reduce: stage 1: SHM intra-node reduce; stage 2: all nodes in group A and C reduce to their counterparts; stage 3: groups B and D reduce to 12 nodes using 3 NICs; stage 4: MPI library reduce to the server node.

B via switch 0. Thus all 24 nodes in A can reduce to their B counterparts in parallel. This requires 0.66 seconds. The same trick is used for reducing from group C to D. The reduced data now resides only on the B and D nodes. The total bandwidth for all 96 nodes in this stage is 4.03 Gb/s.

3. Each node contains 3x100 Mb/s NICs. This allows a node to receive data from three other nodes simultaneously provided the TCP/IP routing tables are correctly configured. We split the 24 nodes in each group into 6 sets of 4 nodes. The first of each set (see node BA in Figure G.1) is designated as the root and the other three nodes send to it via different NICs. This takes 0.9 seconds achieving a bandwidth of 185 Mb/s into each root node, or 2.22 Gb/s across all 12 root nodes.

4. The final step is a standard MPI library reduce from 6 B nodes and 6 D nodes to the master process. This is the slowest step in the process taking 3.16 seconds, including the time spent waiting for the nodes to synchronise since there is some variance in the time taken for the previous steps.

The overall time taken for the optimised reduce to complete is 4.9 seconds. The actual time saved per reduction is 3.6 seconds. The training performance speedup from this saving varies with the duration of the gradient calculation, which depends linearly on the number of training patterns. Figure G.2 illustrates the expected speedup achieved by using the optimised reduce instead of the MPI library reduce, against the

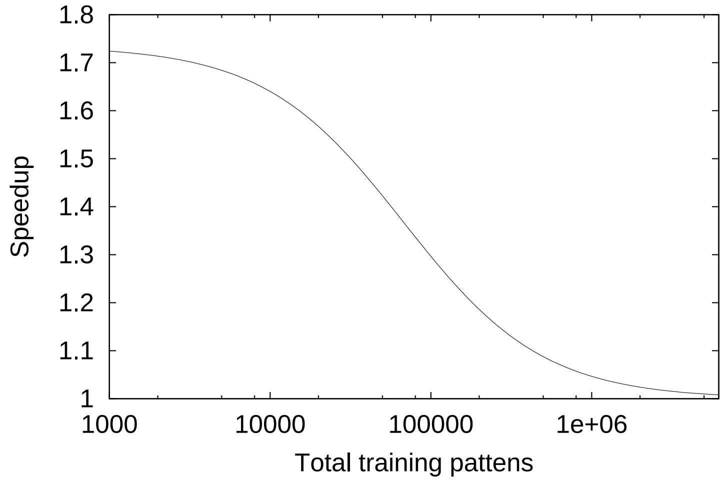


Figure G.2: The theoretical training speedup exhibited after replacing the MPI library reduce with our optimised reduce against the total number of training patters used.

total number of training patterns used. In practice our peak performance of 163.3 GFlops/s benefits by roughly 1% from the optimised reduce, however the speedups are much more marked for smaller (and more frequently encountered) data sets.

Bibliography

Douglas Aberdeen and Jonathan Baxter. General matrix-matrix multiplication using SIMD features of the PIII. In *Euro-Par 2000: Parallel Processing*, Munich, Germany, August 2000. Springer-Verlag.

Douglas Aberdeen and Jonathan Baxter. Emmerald: A fast matrix-matrix multiply using Intel SIMD technology. *Concurrency and Computation: Practice and Experience*, 13:103–119, 2001.

Douglas Aberdeen and Jonathan Baxter. Scaling internal-state policy-gradient methods for POMDPs. In *Proceedings of the 19th International Conference on Machine Learning*, Syndey, Australia, 2002. Morgan Kaufmann. `http://csl.anu.edu.au/~daa/papers.html`.

Douglas Aberdeen, Jonathan Baxter, and Robert Edwards. 92 /MFlop/s, Ultra-Large-Scale Neural-Network training on a PIII cluster. In *Proceedings of Super Computing 2000*, Dallas, TX., November 2000. SC2000 CDROM.

V. M. Aleksandrov, V. I. Sysoyev, and V. V. Shemeneva. Stochastic optimaization. *Engineering Cybernetics*, 5:11–16, 1968.

Timothy R. Anderson. Auditory models with Kohonen SOFM and LVQ for speaker independent phoneme recognition. In *IEEE Internation Conference on Neural Networks*, volume VII, pages 4466–4469, Orlando, Florida, June 1994. IEEE.

Martin Anthony and Peter L. Bartlett. *Neural Network Learning: Theoretical Foundations.* Cambridge University Press, 1999. ISBN 0 521 57353 X.

Howard Anton and Chris Rorres. *Elementary Linear Algebra: applications version.* Wiley, 6th edition, 1991.

Krste Asanović and Nelson Morgan. Experimental determination of precision requirements for back-propagation training of artificial neural networks. Technical report, The International Computer Science Institute, 1991. ftp://ftp.ICSI.Berkeley.EDU/pub/techreports/1991/tr-91-036.ps.gz.

K. J. Åström. Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, 10, 1965.

R. Iris Bahar, Erica A. Frohm, Charles M. Gaona, and Gary D. Hachtel. Algebraic decision diagrams and their applications. In *International Conference of Computer-Aided Design*, pages 188–191. IEEE, 1993.

Leemon C. Baird and Andrew W. Moore. Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 11. MIT Press, 1999.

R. Bakis. Continuous speech recognition via centisecond acoustic states. In *91st Meeting of the Acoustical Society of America*, April 1976.

Bram Bakker. Reinforcement learning with LSTM in non-Markovian tasks with long-term dependencies. Technical report, Leiden University, 2001. `http://www.fsw.leidenuniv.nl/www/w3_func/bbakker/abstracts.htm`.

Peter L. Barlett and Jonathan Baxter. Hebbian synaptic modifications in spiking neurons that learn. Technical report, Computer Sciences Laboratory, RSISE, ANU, November 1999.

Peter L. Barlett and Jonathan Baxter. Estimation and approximation bounds for gradient based reinforcment learning. In *Thirteenth Annual Conference on Computational Learning Theory*, 2000. `http://discus.anu.edu.au/~bartlett/`.

Peter Bartlett and Jonathan Baxter. Estimation and approximation bounds for gradient-based reinforcement learning. *Journal of Computer and System Sciences*, 2002. To appear.

Peter L. Bartlett, Stephane Boucheron, and Gabor Lugosi. Model selection and error estimation. Technical report, Computer Sciences Laboratory, RSISE, ANU, 2000.

Jonathan Baxter. Sketch proof of Exp-GPOMDP convergence. Personal communication, July 2002.

Jonathan Baxter and Peter L. Bartlett. Reinforcement learning in POMDP's via direct gradient ascent. In *Proc. 17th International Conf. on Machine Learning*, pages 41–48. Morgan Kaufmann, San Francisco, CA, 2000.

Jonathan Baxter and Peter L. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

Jonathan Baxter, Peter L. Bartlett, and Lex Weaver. Experiments with infinite-horizon, policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15: 351–381, 2001a.

Jonathan Baxter, Andrew Tridgell, and Lex Weaver. Reinforcement learning and chess. In *Machines that Learn to Play Games*, Advances in Computation: Theory and Practice, chapter 5, pages 91–116. Nova Science Publishers, Huntington, NY, 2001b.

R. Bellman, R. Kalaba, and B. Kotkin. Polynomial approximation — a new computational technique in dynamic programming: allocation processes. *Mathematics of Computation*, 17:155–161, 1963.

Richard Bellman. *Dynamic Programming.* Princeton University Press, Princeton, N.J., 1957.

Yoshua Bengio and Paolo Frasconi. An input output HMM architecture. In *Advances in Neural Information Processing Systems*, volume 7, pages 427–434. The MIT Press, 1995. URL `citeseer.nj.nec.com/bengio95input.html`.

Yoshua Bengio and Paolo Frasconi. Input-output HMM's for sequence processing. *IEEE Transactions on Neural Networks*, 7(5):1231–1249, 1996. URL `citeseer.nj.nec.com/bengio95inputoutput.html`.

Yoshua Bengio, Renato De Mori, Giovanni Flammia, and Ralf Kompe. Global optimization of a neural network-hidden Markov model hybrid. *IEEE Transactions on Neural Networks*, 3(2):252–259, March 1992.

Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dymanic Programming.* Athena Scientific, 1996.

J. Bilmes, K. Asanovic, J. Demmel, D. Lam, and C. W. Chin. PHiPAC: A portable, high-performace, ANSI C coding methodology and its application to matrix multiply. Technical report, University of Tennessee, August 1996. `http://www.icsi.berkeley.edu/~bilmes/phipac`.

Jeff Bilmes, Krste Asanovic, Chee-Whye Chin, and Jim Demmel. Using PHiPAC to speed Error Back-Propogation learning. In *ICASSP*, April 1997.

Jim Blythe. Decision-theoretic planning. *AI Magazine*, 1(20), Summer 1999. `http://www.isi.edu/~blythe/papers/aimag.html`.

Blai Bonet. An ϵ-optimal grid-based algorithm for partially observable Markov decision processes. In *19th International Conference on Machine Learning*, Sydney, Australia, June 2002.

Herv A. Bourlard and Nelson Morgan. *Connectionist Speech Recognition* A Hybrid Approach. Kluwer Academic Publishers, 1st edition, 1994.

Hervé Bourlard, Yochai Konig, and Nelson Morgan. Remap : Recursive estimation and maximization of a posteriori probabilities in connectionist speech recognition. In *Proc. EUROSPEECH '95*, Madrid, September 1995.

Hervé A. Bourlard and Nelson Morgan. *Hybrid HMM/ANN Systems for Speech Recognition: Ovierview and New Research Directions*, volume 1387 of *Lecture Notes in Artificial Intelligence*, pages 389–417. Springer-Verlag, 1998.

C. Boutilier and D. Poole. Computing optimal policies for partially observable decision processes using compact representations, 1996.

Xavier Boyen and Daphne Koller. Tractable inference for complex stochastic processes. In *Conference on Uncertainty in Artificial Intelligence*, pages 33–42, 1998. URL `citeseer.nj.nec.com/boyen98tractable.html`.

Ronen I. Brafman. A heuristic variable grid solution method for POMDPs. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI '97)*, 1997.

Leo Breiman. *Probability.* Addison-Wesley, 1966.

J. S. Bridle and L. Dodd. An alphanet approach to optimising input transformations for continuous speech recognition. In *Proc. International Conference on Acoustics, Speech and Signal Processing*, number 5.7, pages 277–280. IEEE, 1991.

John S. Bridle. ALPHA-NETS: A recurrent 'neural' network architecture with a hidden Markov model interpretation. *Speech Communication*, 9:83–92, February 1990.

John S. Bridle. *Spech Recognition and Understanding. Recent Advances*, chapter Neural Networks or Hidden Markov Models for Automatic Speech Recognition: Is there a Choice?, pages 225–236. Number F75 in NATO ASI. Springer-Verlag Berlin, 1992.

George W. Brown. *Statistical Papers in Honor of George W. Snedecor*, chapter Recursive Sets of Rules in Statistical Decision Processes, pages 59–76. Iowa State University Press, Ames, Iowa, 1972. ISBN 0-8138-1585-1.

George Casella and Christian P. Robert. Rao-Blackwellisation of sampling schemes. *Biometrika*, 83(1):81–94, March 1996.

Anthony Cassandra. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes.* PhD thesis, Brown University, May 1998.

Anthony R. Cassandra. POMDPs for Dummies: POMDPs and their algorithms, sans formula, January 1999. `http://www.cs.brown.edu/research/ai/pomdp/tutorial/index.html`.

Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Articial Intelligence*, 1994.

Hsien-Te Cheng. *Algorithms for Parially Observable Markov Decision Processes.* PhD thesis, University of British Columbia, British Columbia, Canada, 1988.

Lonnie Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *National Conference on Artificial Intelligence*, pages 183–188, 1992.

T. Dean and K. Kanazawa. A model for reasoning about presistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Roayal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

L. Deng. A dynamic feature based approach to the interface between phonology and phonetics for speech modelling and recognition. *Speech Communications*, (24):299–323, 1998.

Thomas G. Dietterich. An overview of MAXQ hierarchical reinforcement learning. In *SARA*, pages 26–44, 2000. URL `citeseer.nj.nec.com/dietterich00overview.html`.

J. J. Dongarra, J. Du Croz, I. S. Duff, and S. Hammarling. Algorithm 679: A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Soft.*, 16:18–28, 1990. `http://www.netlib.org/blas/index.html`.

D. Draper, S. Hanks, and D. Weld. A probabilistic model of action for least-commitment planning with information gathering. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 178–186. Morgan Kaufmann, 1994.

Alain Dutech. Solving POMDPs using selected past events. In *Proceedings of the 14th European Conference on Articial Intelligence, ECAI2000*, pages 281–285, 2000. URL `citeseer.nj.nec.com/dutech00solving.html`.

Alain Dutech, Olivier Buffet, and Francois Charpillet. Multi-agent systems by incremental gradient reinforcement learning. In *Proceedings of the Seventeenth International Joint Conference on Articial Intelligence, IJCAI-01*, pages 833–838, Seattle, WA, 4–10 August 2001. URL `citeseer.nj.nec.com/dutech01multiagent.html`.

Stephen Elliott. *Signal Processing for Active Control*. Academic Press, 2001.

Yaalov Engel and Shie Mannor. Learning embedded maps of Markov processes. In *The Eighteenth International Conference on Machine Learning*, June 2001.

Examiner. Auxillary memory examples. Comments from an anonymous examiners report, January 2003.

Frank Fallside. *Speech Recognition and Understanding. Recent Advances*, chapter Neural Networks for Continuous Speech Recognition, pages 237–257. Number F75 in NATO ASI. Berlin: Springer-Verlag, Cambridge University, 1992.

Paul Fearnhead. *Sequential Monte Carlo methods in filter theory.* PhD thesis, Merton College, University of Oxford, 1998.

Terrence L Fine. *Feedforward Neural Network Methodology.* Springer, New York, 1999.

Sadaoki Furui. An overview of speaker recognition technology. In *ESCA Workshop on Automatic Speaker Recognition, Identification and Verification*, pages 1–9. ESCA, ESCA, 1994.

John S. Garofolo, Lori F. Lamel, William M. Fisher, Jonathan G. Fiscus, David S. Pallett, Nancy L. Dahlgren, and Victor Zue. TIMIT acoustic-phonetic continuous speech corpus, 1993. `http://www.ldc.upenn.edu/Catalog/LDC93S1.html`.

Héctor Geffner and Blai Bonet. Solving large POMDPs by real time dynamic programming. Working Notes Fall AAAI Symposium on POMDPs, 1998. `http://www.cs.ucla.edu/~bonet/`.

Mohammad Ghavamzadeh and Sridhar Mahadevan. Continuous-time hierarchical reinforcement learning. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 186–193. Morgan Kaufmann, June 2001.

Matthew R. Glickman and Katia Sycara. Evolutionary search, stochastic policies with memory, and reinforcement learning with hidden state. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 194–201. Morgan Kaufmann, June 2001.

Peter W Glynn. Stochastic approximation for Monte-Carlo optimization. In *Proceedings of the 1986 Winter Simulation Conference*, pages 356–365, 1986.

Peter W Glynn. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33:75–84, 1990.

Peter W. Glynn. Importance sampling for Monte Carlo estimation of quantiles. Technical report, Dept. of Operations Research, Stanford University, 1996. URL `citeseer.nj.nec.com/glynn96importance.html`.

Peter W Glynn and Paul L'Ecuyer. Likelihood ratio gradient estimation for regenerative stochastic recursions. *Advances in Applied Probability, 27, 4 (1995)*, 27: 1019–1053, 1995.

P. S. Gopalakrishnan, D. Kanevsky, A. Nadas, and D. Nahamoo. A generalization of the Baum algorithm to rational objective functions. In *ICASSP-89*, 1989.

Geoffrey J. Gordon. Reinforcement learning with function approximation converges to a region. In *Proceedings of Neural Information Processing Systems*, volume 13, pages 1040–1046, 2001. URL `citeseer.nj.nec.com/gordon01reinforcement.html`.

Anne Greenbaum. *Iterative Methods for Solving Linear Systems.* Frontiers in applied mathemtics. SIAM, Philadelphia, PA, 1997. ISBN 0-89871-396-X.

Evan Greensmith, Peter L. Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 14, Vancouver, BC, December 2002. MIT Press.

Bruce Greer and Greg Henry. High performance software on Intel Pentium Pro processors or Micro-Ops to TeraFLOPS. Technical report, Intel, August 1997. `http://www.cs.utk.edu/~ghenry/sc97/paper.htm`.

Carlon Guestrin, Daphne Killer, and Ronald Parr. Solving factored POMDPs with linear value functions. In *IJCAI-01 workshop on Palling under Uncertainty and Incomplete Information*, Seattle, Washington, August 2001a. URL `citeseer.nj.nec.com/440372.html`.

Carlos Guestrin, Daphne Koller, and Ronald Parr. Max-norm projections for factored MDPs. In *IJCAI'01*, pages 673–682, Seattle, WA, 2001b. URL `citeseer.nj.nec.com/guestrin01maxnorm.html`.

Eric A. Hansen. Solving POMDPs by searching in policy space. In *The Eighth Conference on Uncertainty in Artificial Intelligence*, pages 211–219, Madison, WI, 1998. URL `citeseer.nj.nec.com/hansen98solving.html`.

Eric A. Hansen and Zhengzhu Feng. Dynamic programming for POMDPs using a factored state representation. In *The Fifth International Conference on Artificial Intelligence Planning and Scheduling*, pages 130–139, Breckenridge, Colarado, April 2000. URL `citeseer.nj.nec.com/hansen00dynamic.html`.

Jean-Paul Haton. Connectionist and hybrid models for automatic speech recognition. In *Proceedings of the NATO Advanced Study Institute on Computational Models of Speech Pattern Processing*, number F169 in NATO ASI, LORIA/Université Henri Poincaré, France, July 1997. Springer-Verlag Berlin.

Thomas Hauser, Timothy I. Mattox, Raymond P. LeBeau, Henry G. Dietz, and P. George Huang. High-cost CFD on a low-cost cluster. In *Proceedings of SC2000 (CD-ROM)*, Dallas, TX., November 4–10 2000.

Milos Hauskrecht. Incremental methods for computing bounds in partially observable Markov decision processes. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, pages 734–739, Providence, Rhode Island, 1997. MIT Press. ISBN 0-262-51095-2.

Milos Hauskrecht. Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 13:33–94, August 2000.

Simon Haykin. *Neural Networks: A comprehensive foundation.* Prentice-Hall, New Jersey, 2nd edition, 1999. ISBN 0-13-273350-1.

Natialia Hernandez-Gardio and Sridhar Mahadevan. Hierarchical memory-based reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 13, 2001.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. URL `citeseer.nj.nec.com/hochreiter95long.html`.

Jesse Hoey, Robert St-Aubin, Alan Hu, and Craig Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of the Fifteenth Conference on Uncertainty in Articial Intelligence*, pages 279–288, 1999. URL `citeseer.nj.nec.com/hoey99spudd.html`.

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

X. D. Huang, K. F. Lee, and A Waibel. Connectionist speaker normalization and its applications to speech recognition. In *IEEE Workshop for Neural Networks for Signal Processing*, New Jersey, October 1991. IEEE.

Qiang Huo and Chorkin Chan. The gradient projection method for the training of hidden Markov models. *Speech Communication*, 13:307–313, May 1993.

Mei-Yuh Hwang and Xuedong Huang. Shared distribution hidden Markov models for speech recognition. *IEEE Transactions on Speech and Audio Processing*, 1(4):414–420, 1993.

Ilse C. F. Ipsen and Carl D. Meyer. The idea behind Krylov methods. *American Mathematical Monthly*, 105(10):889–899, 1998. URL `citeseer.nj.nec.com/135899.html`.

T. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. In *Neural Information Processing Systems 11*, 1998. URL `citeseer.nj.nec.com/jaakkola98exploiting.html`.

Tommi Jaakkola, Satinder P. Singh, and Michael I. Jordan. Reinforcement learning algorithm for partially observable Markov decision problems. In *Advances in Neural Information Processing Systems*, volume 7, pages 345–352. The MIT Press, 1995. URL `citeseer.nj.nec.com/jaakkola95reinforcement.html`.

Bing-Hwang Juang and Shigeru Katagiri. Discrimintative learning for minimum error classification. *IEEE Transactions on Signal Pocessing*, 40(12):3043 – 3054, December 1992.

Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, (4):237–285, May 1996.

Michael J. Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. In *IJCAI*, pages 1324–1231, 1999. URL `citeseer.nj.nec.com/kearns99sparse.html`.

Kee-Eung Kim, Thomas Dean, and Nicolas Meuleau. Approximate solutions to factored Markov decision processes via greedy search in the space of finite state controllers. In *Artificial Intelligence Planning Systems*, pages 323–330, 2000. URL `citeseer.nj.nec.com/392137.html`.

Seung Kim, Joon Hwang, and Chang Lee. Impact locating on aircraft structure using low cost cluster. Denver, Colorado, 2001. Gordon Bell Award price/performance ratio winner. Apparently no paper.

Hajime Kimura and Shigenobu Kobayashi. Reinforcement learning for continuous action using stochastic gradient ascent. In *Intelligent Autonomous Systems (IAS-5)*, pages 288–295, 1998.

Hajime Kimura, Kazuteru Miyazaki, and Shigenobu Kobayashi. Reinforcement learning in POMDPs with function approximation. In *Proc. 14th International Conference on Machine Learning*, pages 152–160. Morgan Kaufmann, 1997.

David Kincaid and Ward Cheney. *Numerical Analysis.* Brooks/Cole Publishers, Pacific Grove, California, 1991. ISBN 0-534-13014-3.

Daphne Koller and Ronald Parr. Policy iteration for factored MDPs, 2000. URL `citeseer.nj.nec.com/koller00policy.html`.

V. Konda and J. Tsitsiklis. Actor-critic algorithms, 2000. URL `citeseer.nj.nec.com/434910.html`.

Mikko Kurimo. Training mixture density HMMs with SOM and LVQ. *Computer Speech and Language*, 11(4):321–343, October 1997. `citeseer.nj.nec.com/kurimo97training.html`.

Harold J. Kushner and Dean S. Clark. *Stochastic Approximation Methods for Constrained and Unconstrained Systems.* Number 26 in Applied Mathematical Sciences. Springer-Verlag, 1978. ISBN 0-387-90341-0.

Ivo Kwee, Marcus Hutter, and Jürgen Schmidhuber. Market-based reinforcement learning in partially observable worlds. In *Proceedings of the 11th International Conference on Artificial Neural Networks.* Springer-Verlag, August 2001. URL `citeseer.nj.nec.com/440786.html`.

John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001. URL `citeseer.nj.nec.com/lafferty01conditional.html`.

LAM Team. Lam/mpi source code v6.3.2, 1999. http://www.mpi.nd.edu/lam/download/.

Pier Luca Lanzi. Solving problems in partially observable environments with classifer systems. Technical Report N. 97.45, Dipartimento di Elettronica e Informazione, Politecnico di Milano, October 1997.

Pier Luca Lanzi. Adaptive agents with reinforcement learning and internal memory. In *The Sixth International Conference on the Simulation of Adaptive Behavior (SAB2000)*, 2000. URL `citeseer.nj.nec.com/346913.html`.

Adam Laud and Gerald DeJong. Reinforcement learning and shaping: Encouraging intended behaviors. In *Proceedings of the 19th International Conference on Machine Learning*, Syndey, Australia, 2002. Morgan Kaufmann.

Kai-Fu Lee. *Large-Vocabulary Speaker-Independant Continuous Speech Recognition: The SPHINX System.* PhD thesis, Computer Science Department, Carnegie-Mellon University, April 1988.

Kai-Fu Lee. *Automatic Speech Recognition, The Development of the SPHINX System.* Kluwer international series in engineering and computer science. SECS 62. Kluwer Academic Publishers, 1989.

Kanungo Lee. UMDHMM V1.02 hidden Markov model software. Software, 2000. `http://www.cfar.umd.edu/~kanungo/software/umdhmm-v1.02.tar`.

Tan Lee, P. C. Ching, and L. W. Chan. An RNN based speech recognition system with discriminative training. In *Proceedings of EuroSpeech-95*, volume 3, pages 1667–70, 1995.

R. Gary Leonard and George Doddington. *TI Digits database.* Texas Instruments, 1982. `http://www.ldc.upenn.edu/Catalog/LDC93S10.html`.

D-T. Lin. *The Adaptive Time-Delay Neural Network: Characterization and Applications to Pattern Recognition, Prediction and Signal Processing.* PhD thesis, Institute for Systems Research, University of Maryland, 1994. `http://www.isr.umd.edu/TechReports/ISR/1994/PhD_94-12/PhD_94-12.phtml`.

Long-Ji Lin and Tom M. Mitchell. Memory approaches to reinforcement learning in non-Markovian domains. Technical Report CS-92-138, Carnegie Mellon, Pittsburgh, PA, 1992. `http://citeseer.nj.nec.com/lin92memory.html`.

Richard P. Lippmann. Review of neural networks for speech recognition. *Neural Computation*, 1(1):1–38, 1989.

Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 362–370, San Francisco, CA, 1995. Morgan Kaufmann. URL `citeseer.nj.nec.com/littman95learning.html`.

A. Ljolje, Y. Ephraim, and L. R. Rabiner. Estimation of hidden Markov model parameters by minimizing empirical error rate. In *Proceedings ICASSP 1990*, volume 2, pages 709–712. IEEE Signal Processing Society, IEEE, April 1990.

John Loch and Satinder Singh. Using eligibility traces to find the best memoryless policy in partially observable Markov decision processes. In *Proc. 15th International Conf. on Machine Learning*, pages 323–331. Morgan Kaufmann, San Francisco, CA, 1998. URL `citeseer.nj.nec.com/loch98using.html`.

W. S. Lovejoy. A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research*, 28(1):47–65, 1991.

Christopher Lusena, Judy Goldsmith, and Martin Mundhenk. Nonapproximability results for partially observable Markov decision processes. *Journal of Artificial Intelligence*, 14:83–103, March 2001.

Weiye Ma and Dirk Van Compernolle. TDNN labeling for a HMM recognizer. In *Proc. International Conference on Acoustics, Speech and Signal Processing*, number 8.3, pages 421–424, Albuquerque, New Mexico, April 1990. IEEE.

O. Madani, S. Hanks, and A. Condon. On the undecidability of probabilistic planning and infinite-horizon paritally observable Markov decision problems. In *Proceedings poof the 16th National Conference of Artifical Intelligence*, pages 541–548, 1999.

R. Makar, S. Mahadevan, and M. Ghavamzadeh. Hierarchical multiagent reinforcement learning. In *Proceedings of the Fifth International Conference on Autonomous Agents*, Montréal, Canada, May 2001. ACM. `citeseer.nj.nec.com/article/makar01hierarchical.html`.

Perter Marbach and John N. Tsitsiklis. Gradient-based optimisation of Markov reward processes: Practical variants. In *38th IEEE Conference on Decisions and Control*, December 1999.

Perter Marbach and John N. Tsitsiklis. Gradient-based optimisation of Markov reward processes: Practical variants. Technical report, Center for Communications Systems Research, University of Cambridge, March 2000.

David A. McAllester and Satinder Singh. Approximate planning for factored POMDPs using belief state simplification. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 409–416, 1999.

Andrew McCallum, Dayne Freitag, and Fernando Pereira. Maximum entropy Markov models for information extraction and segmentation. In *Proceedings of ICML 2000*, 2000. `http://www.cs.cmu.edu/~mccallum/`.

Andrew Kachites McCallum. *Reinformcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, 1996. `http://www.cs.rochester.edu/u/mccallum/phd-thesis/`.

Nicolas Meuleau, Kee-Eung Kim, Leslie Pack Kaelbling, and Anthony R. Cassandra. Solving POMDPs by searching the space of finite policies. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 127–136. Computer Science Dept., Brown University, Morgan Kaufmann, July 1999a.

Nicolas Meuleau, Leonid Pechkin, Leslie P. Kaelbling, and Kee-Eung Kim. Off-policy policy search. Technical report, MIT Artificial Intelligence Laboratory and Brown University, 2000.

Nicolas Meuleau, Leonid Peshkin, and Kee-Eung Kim. Exploration in gradient-based reinforcement learning. AI Memo 2001-003, MIT, April 2001. `http://www.ai.mit.edu/research/publications/2001-publications.shtml`.

Nicolas Meuleau, Leonid Peshkin, Kee-Eung Kim, and Leslie Pack Kaelbling. Learning finite-state controllers for partially observable environments. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*. Computer Science Dept., Brown University, Morgan Kaufmann, July 1999b.

Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

Tom M. Mitchell and Sebastian B. Thrun. Learning analytically and inductively. In *Mind Matters: A Tribute to Allen Newell*, pages 85–110. Lawrence Erlbaum Associates, Inc., Mahwah, New Jersey, 1996. URL `citeseer.nj.nec.com/article/mitchell95learning.html`.

George E. Monahan. A survey of partially observable Markov decision processes: Theory, models, and algorithms. *Management Science*, 28(1):1–16, 1982.

Nelson Morgan. Big dumb neural nets: A working brute force approach to speech recognition. In *IEEE Internation Conference on Neural Networks*, volume VII, pages 4462–4465, Orlando, Florida, June 1994. IEEE.

David E. Moriarty, Alan C. Schultz, and John J. Grefenstette. Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, (11):199–229, August 1999.

B. Müller, J. Reinhardt, and M. T. Strickland. *Neural networks : an introduction.* Physics of neural networks. Springer-Verlag, New York, 2nd edition, 1995.

Hernamm Ney. The use of the maximum likelihood criterion in language modelling. In *Proceedings of the NATO Advanced Study Institute on Computational Models of Speech Pattern Processing*, number F169 in NATO ASI, RWTH Aachen – University of Technology, July 1997. Springer-Verlag Berlin.

A. Ng and M. Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *UAI, 2000*, 2000. URL `citeseer.nj.nec.com/297555.html`.

Lee. T. Niles and Harvey. F. Silverman. Combining hidden Markov models and neural network classifiers. In *Proc. International Conference on Acoustics, Speech and Signal Processing*, number 8.2, pages 417–420, Albuquerque, New Mexico, April 1990.

N. J. Nilsson. *Principles of Artificial Intelligence.* Tioga Publishing Company, 1980.

Albino Nogueiras-Rodriíguez, José B. Mari no, and Enric Monte. An adaptive gradient-search based algorithm for discriminative training of HMMs. In *Proceedings of ICLSP'98.* Universitat Politècnica de Catalunya, Causal Productions, 1998.

Yves Normandin. *Hidden Markov models, maximum mutual information estimation and the speech recognition problem.* PhD thesis, McGill University, 1991.

Yves Normandin and Reǵis Cardin. *Developments in High-Performance Connected Digit Recognition*, volume F.75 of *NATO ASI Series*, pages 89–94. Springer–Verlag, Berlin, 1992.

I. Nourbakhsh, R. Powers, and S. Birchfield. DERVISH an office-navigating robot. *AI Magazine*, 16(2):53–60, 1995.

Katsuhiko Ogata. *Modern Control Engineering.* Prentice-Hall, New Jersey, U.S., 2nd edition, 1990. ISBN 0-12-589128-0.

Luis E. Ortiz and Leslie Pack Kaelbling. Adaptive importance sampling for estimation in structured domains. In *Proceedings of the Sixteenth Annual Conference on Uncertainty in Articial Intelligence (UAI2000)*, pages 446–454. Morgan Kaufmann Publishers, 2000.

Christos H. Papadimitriou and John N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

Ronald Parr and Stuart Russell. Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1088–1094. Morgan Kaufmann, 1995. URL `citeseer.nj.nec.com/parr95approximating.html`.

D. B. Paul. Speech recognition using hidden Markov models. *The Lincoln Laboratory Journal*, 1:41–62, 1990.

Mark D. Pendrith and Michael J. McGarity. An analysis of direct reinforcement learning in non-Markovian domains. In *Proc. 15th International Conf. on Machine Learning*, pages 421–429. Morgan Kaufmann, San Francisco, CA, 1998. URL `citeseer.nj.nec.com/11788.html`.

Leonid Peshkin. *Policy Search for Reinforcement Learning.* PhD thesis, Brown University, 2002. Draft.

Leonid Peshkin, Nicolas Meuleau, and Leslie Kaelbling. Learning policies with external memory. In *Proceedings of the Sixteenth International Conference in Machine Learning*, pages 307–314. Morgan Kaufmann, 1999.

Leonid Peshkin and Christian R. Shelton. Learning from scarce experience. In *Proceedings of ICML*, number 19, Sydney, Australia, 2002. `http://www.ai.mit.edu/~pesha/Public/papers.html`.

Leonid M. Peshkin. Thesis proposal: Architectures for policy search. `http://www.ai.mit.edu/~pesha/Public/papers.html`, July 2000.

Stephen M. Pollock. A simple model of search for a moving target. *Operations Research*, 18(5):883–903, 1970.

Louis C. W. Pols. Psycho-acoustics and speech perception. In *Proceedings of the NATO Advanced Study Institute on Computational Models of Speech Pattern Processing*, number F169 in NATO ASI, IFOTT, University of Amsterdam, July 1997. Springer-Verlag Berlin.

Alan B. Poritz. Hidden Markov models: A guided tour. In *ICASSP '88*, pages 7–13. Morgan Kaufmann, 1988.

Pascal Poupart and Craig Boutilier. Value-directed belief state approximation for POMDPs. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 409–416, 2000. `http://www.cs.toronto.edu/~cebly/papers.html`.

Pascal Poupart and Craig Boutilier. Vector-space analysis of belief-state approximation for POMDPs. In *Uncertainty in Artificial Intelligence 2001*, August 2001.

Pascal Poupart, Luis E. Ortiz, and Craig Boutilier. Value-directed sampling methods for monitoring POMDPs. In *Uncertainty in Artificial Intelligence 2001*, August 2001. URL `citeseer.nj.nec.com/445996.html`.

Doina Precup. *Temporal Abstraction in Reinforcement Learning.* PhD thesis, University of Massachusetts, may 2000. URL `citeseer.nj.nec.com/precup00temporal.html`.

John G. Proakis. *Digital Communications.* McGraw-Hill, 3rd edition, 1995. ISBN 0-07-051726-6.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley series in probability and mathematical statistics. Wiley, 1994. ISBN 0-471-61977-9.

L. R. Rabiner and B. H. Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, pages 4–16, January 1986.

Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech processing. In *Proceedings of the IEEE*, volume 77. IEEE, February 1989.

S. Ran and J. B. Millar. Phoneme discrimination using hierarchically organised connectionist networks. In *Proceedings of Second Australian Conference of Neural Networks*, pages 279–282, Sydney, Australia, 4–6 February 1991.

S. Ran and J. B. Millar. Two schemes of phonetic feature extraction using artificial neural networks. In *Proceedings of Eurospeech'93*, pages 1607–1610, Berlin, Germany, 1993.

W. Reichl and G. Ruske. Discriminative training for continuous speech recognition. In *Europ. Conf. on Speech Communication and Technology*, volume 1, pages 537–540, September 1995.

M I Reiman and A Weiss. Sensitivity analysis via likelihood ratios. In *Proceedings of the 1986 Winter Simulation Conference*, 1986.

M I Reiman and A Weiss. Sensitivity analysis for simulations via likelihood ratios. *Operations Research*, 37, 1989.

Michael D. Richard and Richard P. Lippmann. Neural network classifiers estimate bayesian *a posteriori* probabilities. *Neural Computation*, 3(4):461–483, Winter 1991.

Tony Robinson. An application of recurrent nets to phone probability estimation. *IEEE Transactions on Neural Networks*, 5(3):298–305, 1994.

Andrés Rodríguez, Ronald Parr, and Daphne Koller. Reinforcement learning using approximate belief states. In *Advances in Neural Information Processing Systems*, volume 12, 2000.

Dana Ron, Yoram Singer, and Naftali Tishby. The power of amnesia. In *Advances in Neural Information Processing Systems*, volume 6, pages 176–183. Morgan Kaufmann Publishers, Inc., 1994. URL `citeseer.nj.nec.com/ron94power.html`.

Nicholas Roy and Sebastian Thrun. Integrating value functions and policy search for continuous Markov decision processes. In *Advances in Neural Information Processing Systems*, volume 14, 2001.

Reuven Y. Rubinstein. *Some Problems in Monte Carlo Optimization.* PhD thesis, 1969.

D. Rumelhart, G. Hinton, and R. R. Williams. *Parallel Distributed Processing*, chapter Learning internal representations by error propagation. MIT Press, Cambridge, MA., 1986.

Brian Sallans. Learning factored representations for partially observable Markov decision processes. In *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 2000. URL `citeseer.nj.nec.com/sallans00learning.html`.

Rafał Sałustowicz and Jürgen Schmidhuber. Probabilistic incremental program evolution. *Evolutionary Computation*, 5(2):123–141, 1997.

Lawrence K. Saul and Mazin G. Rahim. Markov processes on curves. Technical report, 1997.

J. H. Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 2(4):234–242, 1992.

Jürgen Schmidhuber. Reinforcement learning in Markovian and non-Markovian environments. In *Advances in Neural Information Processing Systems*, volume 3, pages 500–506. Morgan Kaufmann Publishers, Inc., 1991. URL `citeseer.nj.nec.com/100953.html`.

Michael Schuster. *On supervised learning from sequential data with applications for speech recognition.* PhD thesis, Graduate School of Information Science, Nara Institute of Science and Technology, February 1999.

Mike Schuster. Bi-directional recurrent neural networks for speech recognition. Technical report, 1996.

Mike Schuster. *Encyclopedia of Electrical and Electronics Engineering*, chapter Neural networks for speech processing. John Wiley & Sons, June 1998.

Kristie Seymore, Stanley Chen, Sam-Joo Doh, Maxine Eskenaziand Evandro Gouvea, Bhiksha Raj, Mosur Ravishankar, Ronald Rosenfeld, Matthew Siegler, Richard Stern ane, and Eric Thayer. The 1997 CMU sphinx-3 english broadcast news transcription system. In *Proceedings of the 1998 DARPA Speech Recognition Workshop.* DARPA, 1998.

Christian Shelton. *Importance Sampling for Reinforcement Learning with Multiple Objectives.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, August 2001a. `http://robotics.stanford.edu/~cshelton/papers/`.

Christian R. Shelton. Policy improvement for POMDPs using normalized importance sampling. In *Uncertainty in Artificial Intelligence*, August 2001b.

Christian R. Shelton. Policy improvment for POMDPs using normalized importance sampling. Technical Report AI Memo 2001-002, MIT, Cambridge, MA, March 2001c. `http://www.ai.mit.edu/people/cshelton/papers/`.

Reid Simmons and Sven Koenig. Probabilistic robot navigation in partially observable environments. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1080–1087, 1995. URL `citeseer.nj.nec.com/article/simmons95probabilistic.html`.

S. Singh, T. Jaakkola, and M. Jordan. Learning without state-estimation in partially observable Markovian decision processes. In *Proceedings of ICML 1994*, number 11, 1994.

Satinder P. Singh, Tommi Jaakkola, and Michael I. Jordan. Reinforcement learning with soft state aggregation. In *Advances in Neural Information Processing Systems*, volume 7, pages 361–368. The MIT Press, 1995. URL `citeseer.nj.nec.com/article/singh95reinforcement.html`.

R D Smallwood and Edward J Sondik. The optimal control of partially observable Markov decision processes over a finite horizon. *Operations Research*, 21(5):1071–1088, 1973.

Richard D. Smallwood, Edward J. Sondik, and Fred L. Offensend. Toward an integrated methodology for the analysis of health-care systems. *Operations Research*, 19(6): 1300–1322, October 1971.

Edward J. Sondik. *The Optimal Control of Paritally Observable Markov Decision Processes.* PhD thesis, Stanford University, Standford, CA., 1971.

Edward J Sondik. The optimal control of partially observable Markov decision processes over the infinite horizon: Discounted costs. *Operations Research*, 26(2):282–304, 1978.

S. N. Srihari, G. Srikantan, T. Hong, and S. W. Lam. *Research in Japanese OCR: Handbook on Optical Character Recognition and Document Image Analysis.* World Scientific Publishing Company, 1996. `http://www.cedar.buffalo.edu/JOCR/`.

V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

Malcom J. A. Strens and Andrew W. Moore. Direct policy search using paired statistical tests. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 545–552. Morgan Kaufmann, June 2001.

N. Suematsu and A. Hayashi. A reinforcement learning algorithm in partially observable environments using short-term memory. In *Advances in Neural Information Processing Systems*, volume 11, pages 1059–1065, 1999. URL `citeseer.nj.nec.com/suematsu99reinforcement.html`.

Richard S. Sutton. Open theoretical questions in reinforcement learning. In *European Conference on Computational Learning Theory*, pages 11–17, 1999. URL `citeseer.nj.nec.com/sutton99open.html`.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge MA, 1998. ISBN 0-262-19398-1.

Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, 12:1057–1063, 2000.

Richard S. Sutton, Satinder Singh, Doina Precup, and Balaraman Ravindran. Improved switching among temporally abstract actions. In *Advances in Neural Information Processing Systems*. MIT Press, 1999.

Nigel Tao, Jonathan Baxter, and Lex Weaver. A multi-agent, policy-gradient approach to network routing. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 553–560. Morgan Kaufmann, June 2001.

Gerald Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6:215–219, 1994.

G Tesuro. Neurogammon: A neural-network backgammon program. In *International Joint Conference on Neural Networks*, pages C33–40, New York, 1990. IEEE.

Georgios Theocharous, Khashayar Rohanimanesh, and Sridhar Mahadevan. Learning and planning with hierarchical stochastic models for robot navigation. In *ICML 2000 Workshop on Machine Learning of Spatial Knowledge*, Stanford, 2000.

Sylvie Thiébaux, Froduald Kabanza, and John Slaney. Anytime state-based solution methods for decision processes with non-Markovian rewards. In *Proceedings of Uncertainty in Artificial Intelligence*, University of Alberta, Edmonton, Canada, August 1–4 2002. Pre-print.

Mithuna Thottethodi, Siddhartha Chatterjee, and Alvin R. Lebeck. Tuning Strassen's matrix multiplication for memory efficiency. In *Proceedings of Super Computing '98*, November 1998. /cdrom/sc98/sc98/techpape/sc98full/thotteth/index.htm.

Sebastian Thrun. Monte Carlo POMDPs. In *Advances in Neural Information Processing Systems 12*. MIT Press, 2000. `http://citeseer.nj.nec.com/thrun99monte.html`.

A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Infomation Theory*, 13(2):260–269, April 1967.

Alex Waibel. Modular construction of time-delay neural networks for speech recognition. *Neural Computation*, 1(1):39–46, 1989.

Narada Dilp Warakagoda. A hybrid ANN-HMM ASR system with NN based adaptive preprocessing. Master's thesis, Institutt for Teleteknikk, Transmisjonsteknikk, May 1996. `http://jedlik.phy.bme.hu/~gerjanos/HMM/hoved.html`.

R. Washington. BI-POMDP: Bounded incremental partially-observable Markov-model planning. In *Proceedings of the Fourth European Conference on Planning*, Toulouse, France, September 1997. Springer-Verlag.

Lex Weaver and Nigel Tao. The optimal reward baseline for gradient-based reinforcement learning. In *Uncertainty in Artificial Intelligence: Proceedings of the Seventeenth Conference (2001)*, pages 538–545, Seattle, WA, 2–5 August 2001. Morgan Kaufman.

Wei Wei and Sarel Vuuren. Improved neural network training of inter-word context units for connected digit recognition. In *ICASSP'98*, pages 497–500, Seattle, WA, May 1998. IEEE.

R. Clint Whaley and Jack J. Dongarra. Automatically tuned linear algebra software. Technical report, Computer Science Department, University of Tennessee, 1997. `http://www.netlib.org/utk/projects/atlas/`.

R. Clint Whaley, Antoine Petitet, and Jack J. Dongarra. Automated empirical optimizations of software and the ATLAS project. Technical report, Dept. of Computer Sciences, Univ. of TN, Knoxville, March 2000. `http://www.cs.utk.edu/~rwhaley/ATLAS/atlas.html`.

John K. Williams and Satinder Singh. Experimental results on learning stochastic memoryless policies for partially observable Markov decision processes. In *Advances in Neural Information Processing Systems*, volume 11, 1999.

R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural network. *Neural Computation*, 1(2):270–280, 1989.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

Phil Woodland. *Hidden Markov Model Toolkit V3.* University of Cambridge, 2001. `http://htk.eng.cam.ac.uk/`.

S. J. Young. Competitive training in hidden Markov models. In *Proc. International Conference on Acoustics, Speech and Signal Processing*, number 13.1, pages 681–684, Albuquerque, New Mexico, April 1990. IEEE.

D. Zhang and J. B. Millar. Digit-specific feature extraction for multi-speaker isolated digit recognition using neural networks. In *Proceedings of 5th Australian International Conference on Speech Science and Technology*, pages 522–527, Brisbane, Australia, 6–8 December 1994.

Nevin L. Zhang. Efficient planning in stochastic domains through exploiting problem characteristics. Technical Report HKUST-CS95-40, Dept. of Computer Science, Hong Kong University of Science and Technology, August 1995.

Nevin L. Zhang and Wenju Liu. Planning in stochastic domains: Problem characteristics and approximation. Technical Report HKUST-CS96-31, Dept. of Computer Science, Hong Kong University of Science and Technology, 1996. URL `citeseer.nj.nec.com/article/zhang96planning.html`.

Nevin L. Zhang and Weihong Zhang. Speeding up the convergence of value iteration in partially observable Markov decision processes. *Journal of Artificial Intelligence*, 14:29–51, Febrauary 2001.

Valentina Bayer Zubeck and Thomas G. Dietterich. A POMDP approximation algorithm that anticipates the need to observe. In *Pacific Rim International Conference on Artificial Intelligence*, pages 521–532, 2000. URL `citeseer.nj.nec.com/bayer00pomdp.html`.