Learning to Use Memory

by

Nicholas Andrew Gorski

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2012

Rackham at Night

Photograph ©Jens Wessling.

For Carly.

ACKNOWLEDGEMENTS

I'm very fortunate to have had the guidance of my advisor, John, as well as that of the rest of my committee. Without their collective thoughts, feedback and input this thesis would not have been possible.

The Soar group, broader Soar community, and other students in the AI lab have had a significant impact on my development as a scientist. In particular, conversations with Erik, Joseph and Justin have shaped my outlook on artificial intelligence as well as contributed directly to this dissertation.

The support of my family has been invaluable to me on a personal level, and without them I can't imagine having even contemplated starting graduate school much less finishing.

TABLE OF CONTENTS

DEDICATION ii

ACKNOWLEDGEMENTS iii

LIST OF FIGURES vii

LIST OF TABLES xi

ABSTRACT xii

CHAPTER

I. Introduction 1

II. Background 5

2.1 Reinforcement Learning 5
2.2 Working Memory 10
2.3 Actions to Use Memory 11
2.4 Prior Work Learning to Use Memory 12
2.5 Learning to Use Episodic Memory 15

III. A Framework to Explore the Dynamics of Memory and Task 21

3.1 A Conceptual Framework for Learning to Use Memory 21
3.2 Reinforcement Learning in Our Framework 25
3.3 Memory Models 26
3.3.1 Bit Memory 27
3.3.2 Gated Memory 30
3.3.3 Brief Discussion of Abstract and Concrete Knowledge 32
3.4 TMaze 33
3.5 Task Dimensions and Parameterized TMazes 35

IV. Experimental and Analytical Methodologies 37

4.1 Methodology for Evaluating Learning Performances 37
4.2 Analysis Techniques 38
4.2.1 Individual Behavior Analysis 39
4.2.2 State Diagram Analysis 39
4.3 Parameter Selection Methodology 41

V. Base TMaze 44

5.1 Bit and Gated Memory in the Base TMaze 44
5.1.1 Individual Behavior Analysis 46
5.1.2 State Diagram Analysis 48
5.2 Architectural Modifications to Bit Memory 51
5.2.1 Discussion 56
5.2.2 Additional Architectural Modifications to Bit Memory 58
5.2.3 Unmodified Memory Models with Eligibility Traces 58

VI. Temporal Delay 62

6.1 Description of Temporal Delay 62
6.2 Temporal Delay Experiments 64
6.2.1 Unique Observations in TMaze Extended with Temporal Delay 64
6.2.2 Identical Observations in TMaze Extended with Temporal Delay 70
6.3 Summary of Temporal Delay Experiments 79

VII. Number of Dependent Decisions 82

7.1 Description of Number of Dependent Decisions 82
7.2 Number of Dependent Decisions Experiments 84
7.2.1 Number of Dependent Decisions with Unique Observations 84
7.2.2 Dependent Decisions with Identical Observations and Agents with Gated Memory 89
7.2.3 Dependent Decisions with Identical Observations and Agents with Bit Memory 98

VIII. Number of Distinct Symbols 100

8.1 Description of Number of Distinct Symbols 100
8.2 Number of Distinct Symbols Experiments 101

IX. Concurrent Knowledge 107

9.1 Description of Concurrent Knowledge 107
9.2 Concurrent Knowledge Experiments 109

X. Second-Order Knowledge 116

10.1 Description of Second-Order Knowledge 116
10.2 Summary of Experiments 119

XI. Discussion 121

11.1 Discussion of Results Along Dimensions of Task 121
11.2 Contrasting Gated and Bit Memory Models 126
11.3 Evaluating Online Learning Performance 130
11.4 Generality of Results 131
11.5 Eligibility Traces 132
11.6 On Numerical Parameters 134
11.7 Implications for Cognitive Architectures 135
11.8 The Future of Learning To Use Memory 137
11.9 Contributions 138
11.10Future Work 140

BIBLIOGRAPHY 143

LIST OF FIGURES

Figure

2.1 The paradigmatic reinforcement learning setting. 6

2.2 Performances of agents learning to use bit memory in Well World for various amounts of a priori procedural knowledge. 18

3.1 The paradigmatic reinforcement learning setting. 22

3.2 A conceptual framework endowing a reinforcement learning agent with internal memory. 22

3.3 Possible states of bit memory and how the toggle action transitions between them. 28

3.4 (a) Bit memory consisting of a single bit of knowledge. (b) Bit memory consisting of more than one bit of knowledge, where each conceptual memory slot stores a discrete bit. 30

3.5 The base TMaze task. 34

4.1 State diagram of a completely observable TMaze task. 41

4.2 State diagram of a partially observable TMaze task. 42

5.1 Performances of agents endowed with bit and gated memory in base TMaze task. 45

5.2 State diagram for an agent learning to use gated memory in the TMaze task. 49

5.3 State diagram for an agent learning to use bit memory in the TMaze task. 50

5.4 State diagram for an agent learning to use bit memory in the TMaze task, where the framework has been modified to prevent internal toggle actions in the choice location. 52

5.5 State diagram for an agent in the TMaze task learning to use null-initialized bit memory with set actions. 53

5.6 Restricting the bit memory toggle action and initializing bit memory to null. 54

5.7 Agents learning to use bit memory and gated memory in the base TMaze task with eligibility traces. 59

6.1 The TMaze task extended along the dimension of temporal delay. . 63

6.2 Agent performance when learning to use gated memory in the temporal delay TMaze task with unique observations, for zero to forty intermediate locations. 65

6.3 Number of episodes until convergence to optimal behavior for agents learning to use gated memory in the TMaze extended along the dimension of temporal delay. 66

6.4 Agent performance when learning to use bit memory in the temporal delay TMaze task with unique observations, for zero to ten intermediate locations, and an eligibility trace decay rate of 0.9, normalized to present the difference from optimal behavior. 68

6.5 Agent performance when learning to use gated memory in the temporal delay TMaze task with identical observations, for zero to ten intermediate locations. 71

6.6 Agent performance when learning to use gated memory in the temporal delay TMaze task with identical observations, for zero to ten intermediate locations and an eligibility trace decay rate of 0.75. . . 74

6.7 Agent performance when learning to use gated memory in the temporal delay TMaze task with identical observations, for zero to ten intermediate locations, when internal actions in the choice location are inhibited. 75

6.8 Agent performance when learning to use bit memory in the temporal delay TMaze task with identical observations, for zero to ten intermediate locations. 76

6.9 Agent performance when learning to use bit memory in the temporal delay TMaze task with identical observations, for zero to ten intermediate locations, in conjunction with eligibility traces and an eligibility trace decay rate of 0.9. 78

7.1 The TMaze task extended along the dimension of number of dependent decisions, for a total of two dependent decisions. 83

7.2 Agent performance when learning to use gated memory in the TMaze task extended with additional dependent actions and unique observations, for one to ten dependent actions. 85

7.3 Number of episodes until convergence to optimal behavior for agents learning to use gated memory in TMaze tasks extended with additional dependent decisions. 86

7.4 State diagram for an agent endowed with gated memory performing the TMaze task with two dependent choices and unique observations. 87

7.5 Agent performance when learning to use bit memory with eligibility traces in the TMaze task extended with additional dependent actions and unique observations, for one to eight dependent actions and an eligibility trace decay rate of 0.7. 88

7.6 Agent performance when learning to use gated memory in the TMaze task extended with additional dependent actions and identical observations, for one to ten dependent actions. 90

7.7 Agent performance when learning to use gated memory with eligibility traces in the TMaze task extended with additional dependent actions and identical observations, for one to ten dependent actions and an eligibility trace decay rate of 0.7. 91

7.8 State diagram for an agent endowed with gated memory performing the TMaze task with two dependent choices and identical observations. 92

7.9 The portion of the state diagram that is of particular interest for an agent endowed with gated memory performing in a TMaze task with two dependent decisions and identical observations. 94

7.10 A POMDP that captures aspects of the state diagram for an agent with gated memory in TMazes extended with additional decisions and identical observations. 95

7.11 POMDP from Figure 7.10 extended with an additional action in state 0 which results in terminal reward of 0.0. 96

7.12 Mean agent performances in both POMDP tasks, with and without eligibility traces. 97

7.13 Agent performance when learning to use bit memory with eligibility traces in the TMaze task extended with additional dependent actions and identical observations, for one to eight dependent actions and an eligibility trace decay rate of 0.7. 99

8.1 The TMaze task extended along the dimension of the amount of salient knowledge 101

8.2 Agent performance when learning to use gated memory in TMaze tasks with varying numbers of distinct symbols, for thirty to two hundred distinct symbols and corresponding actions. 102

8.3 Amount of experience required for agents learning to use gated memory in TMaze tasks with various numbers of distinct symbols to converge to optimal behavior, for two to one hundred forty distinct symbols and corresponding actions. 103

8.4 Agent performance when learning to use bit memory with eligibility traces in TMaze tasks with varying numbers of distinct symbols, for two to eight distinct symbols and corresponding actions. 105

9.1 The TMaze task extended with concurrent knowledge. 108

9.2 Mean agent performance when learning to use gated memory with eligibility traces in TMaze tasks with concurrent knowledge, for one to eight concurrent salient symbols. 110

9.3 Mean agent performances when learning to use bit memory with eligibility traces in TMaze tasks with concurrent knowledge, for one to four concurrent salient symbols. 112

9.4 Mean agent performances when learning to use bit memory with eligibility traces in the base TMaze task, for one to four slots of bit memory. 113

10.1 The TMaze task extended with second-order knowledge. 117

LIST OF TABLES

Table

5.1 Summary of individual analysis of behaviors for agents endowed with null-initialized bit memory in the base TMaze task. 55

6.1 Summary of individual analysis of behaviors for agents learning to use bit memory in the temporally extended TMaze task with unique observations. 67

6.2 Summary of individual analysis of behaviors for agents learning to use gated memory in the temporally extended TMaze task with identical observations. 72

6.3 Summary of individual analysis of behaviors for agents learning to use bit memory in the temporally extended TMaze task with identical observations. 77

6.4 Summary of results in TMazes extended along the temporal delay dimension. 81

11.1 Summary of experiments presented in Chapter IV. 122

11.2 Summary of empirical results and how performance scales along task dimensions in the TMaze domain. 123

ABSTRACT

Learning to Use Memory

by

Nicholas Andrew Gorski

Chair: John E Laird

This thesis is a comprehensive empirical exploration of using reinforcement learning to learn to use simple forms of working memory. Learning to use memory involves learning how to behave in the environment while simultaneously learning when to select internal actions that control how knowledge persists in memory and learning how to use that information stored in memory to make decisions. We focus on two different models of memory: bit memory and gated memory. Bit memory is inspired by prior reinforcement learning literature and stores abstract values, which an agent can learn to associate with task history. Gated memory is inspired by human working memory and stores perceptually grounded symbols. Our goal is to determine computational bounds on the tractability of learning to use these memories. We conduct a comprehensive empirical exploration of the dynamics of learning to use memory models by modifying a simple partially observable task, TMaze, along specific dimensions: length of temporal delay, number of dependent decisions, number of distinct symbols, quantity of concurrent knowledge, and availability of second-order knowledge. We find that learning to use gated memory is significantly more tractable than learning to use bit memory because it stores perceptually grounded symbols in memory. We further find that learning performance scales more favorably along temporal delay, distinct symbols, and concurrent knowledge when learning to use gated memory than along other dimensions. We also identify situations in which agents fail to learn to use gated memory optimally which involve repeated identical observations which result in no unambiguous trajectories through the underlying task and memory state space.

CHAPTER I

Introduction

This thesis explores artificial agents that learn to use memory. We conduct this study in the context of artificial reinforcement learning agents that learn to perform in environments in which reactive behaviors depending on immediate perception are insufficient to perform optimally. Agents thus require internal memory to maintain knowledge about parts of the environment that they perceived in the past but can no longer immediately perceive. Control of these internal memories is achieved by taking memory actions, which from the agent's perspective are the same as taking actions in the environment save for their functional effects. Agents must also learn to use the knowledge that they store in memory. Learning to use memory, then, involves two learning problems that must be learned simultaneously: learning how and when to take memory actions, as well as learning how to make use of knowledge that is contained in memory.

Our exploration of learning to use memory is empirical. In this thesis we identify a base task, the TMaze (described in Chapter III), and extend it by parameterizing it along dimensions of task that relate to how memory must be used in order to perform the task successfully. We use the TMaze task because it is the simplest task formulation in which an agent must use historical information to direct its behavior so that it can receive the highest possible reward.

We investigate agents that learn to use two working memory models in this work: one that stores abstract knowledge, and another that stores perceptually grounded knowledge (both described in detail in Chapters II and III). As we found in related work (Chapter II), while learning to use memory has been explored in isolated tasks, no comprehensive study of learning to use memory has yet been undertaken. Learning to use memory in these isolated tasks has focused on learning to use working memory models; as such they are a natural starting point for a comprehensive exploration of learning to use memory.

We explore memory and task combinations in the reinforcement learning setting, which allows us to objectively measure agent behavior. We are specifically concerned with online agents that learn by taking actions that directly interact with their environments and memory; agents that use models of their environments to simulate the effects of their actions before taking them are beyond the scope of this work. We restrict our exploration to the foundational online temporal difference reinforcement learning algorithms.

The goals of this work are to: formulate the problem of learning to use memory given specific constraints of a task and cognitive architecture; understand how learning performance scales along task dimensions that relate to the use of memory; determine whether there are differences in the performances afforded by different memory mechanisms; and develop empirical bounds that dictate when learning to use memory is tractable in the reinforcement learning setting. Identifying computational bounds over learning to use memory is a first step in investigating which portions of memory control can be learned through direct experience with a task and memory mechanism, and which portions must be provided to an agent through other means.

Learning to use memory while simultaneously learning to perform a task is a central aspect of our work. When learning both simultaneously, an agent's behavior in the task affects how information can flow to memory. Similarly, an agent's behavior over memory affects how it can use knowledge in memory to condition behavior in the task. The implications of this duality will be discussed throughout this thesis, and formulate the central problem of learning to use memory.

All of the tasks that we explore, including the base TMaze task, are *partially observable*. We discuss partial observability in more detail in Chapter II, but mention here that in partially observable tasks an agent cannot observe the complete state of the environment. Using a memory mechanism to explicitly store knowledge of past experience, as in our approach, is one way that an agent can overcome partial observability.

After identifying trends in how mean learning performances scale along dimensions of task, as well as differences in learning performances afforded by gated and bit memory, we then investigate the behaviors exhibited by individual agents in order to determine why particular combinations of task and memory model present difficulties for learning agents (Section 4.2.1). We also construct state diagrams in order to more formally characterize what aspects of the task and memory model result in different types of learning performances (Section 4.2.2).

In certain cases, and particularly in the case of the base TMaze task, we modify the working memory mechanisms to determine the specific aspects of memory that help to support an agent's successful task learning.

We opted to perform an empirical evaluation of learning to use memory instead of other types of evaluations, such as subjective evaluation and theoretical evaluations. Whiteson & Littman (2011) provide an excellent discussion over the merits of these three types of evaluations as applied to reinforcement learning. Our motivation was pragmatic: performing both an empirical and theoretical evaluation of learning to use memory was beyond the scope of this thesis, and as literature lacked a comprehensive empirical evaluation of learning to use memory it was a natural first step.

In particular, while empirical evaluation of reinforcement learning algorithms is increasingly gaining acceptance (for example, see the recent special issue of *Machine Learning* vol. 84:1-2 dedicated to the empirical evaluation of reinforcement learning), we are interested not in the relative performance of algorithms on a core set of benchmark tasks but rather the underlying relationships of reinforcement learning and an agent's deliberate control of internal memory. This is similar to many of the empirical evaluations of the canonical reinforcement learning textbook (Sutton & Barto, 1998), in which relationships between parameters and task are shown and discussed not by closed-form theoretical analysis but rather by running agents across parameter settings and visualizing the resulting performances. We are agnostic to the choice of specific algorithms, and instead focus on how empirical performances scale along relevant dimensions of task; an empirical analysis along those dimensions is the fastest and most practical course towards achieving an understanding.

Note that while the foundation of this thesis is its empirical evaluation, the state diagram analysis that we perform is a theoretical analysis. However, it does not provide bounds and closed form equations, but rather supplements the empirical evaluation in order to confirm our understanding of why empirical phenomena arise.

While our evaluations of agent learning performances are driven by understanding whether agents can learn to perform a task optimally while learning to use memory, our goal is not to develop agent frameworks that are guaranteed to learn on classes of tasks optimally. We do not propose new algorithms that improve agent performance when learning to use memory. Instead, we seek fundamental relationships, through empirical studies, that identify in which tasks agents can and cannot learn to use memory, and the reasons why.

CHAPTER II

Background

In this chapter we present background on reinforcement learning and memory with the intent of defining terms and general concepts that are used throughout the rest of this thesis. For a more detailed overview of reinforcement learning concepts, we recommend that an interested reader consult Sutton & Barto (1998).

2.1 Reinforcement Learning

Reinforcement learning approaches are defined by a class of problems that they solve, rather than by specific method of problem solving. In a reinforcement learning task, an agent takes actions in an environment, observes the environment and receives a reward. The goal of a reinforcement learning agent is to maximize the reward that it receives.

The reinforcement learning paradigm is general. For example, the familiar game of chess can be represented as a reinforcement learning problem: an agent takes actions by moving a playing piece, looks at the board to observe the configuration of the pieces, and could receive a positive reward for winning the game and a negative reward for losing. In general, any task in which a specific behavior is desired can be represented as a reinforcement learning problem by rewarding an agent when it exhibits that behavior.

Figure 2.1 illustrates the reinforcement learning setting, in which an agent selects an action in the environment and in return receives an observation and a reward. The environment has state, and an action changes the state of the environment according to a stochastic transition function. After an agent selects an action and the environment state is updated, an agent receives an observation and reward. The observation[1] is a function of the current environment state; the reward is a function of the previous state, selected action, and current state.

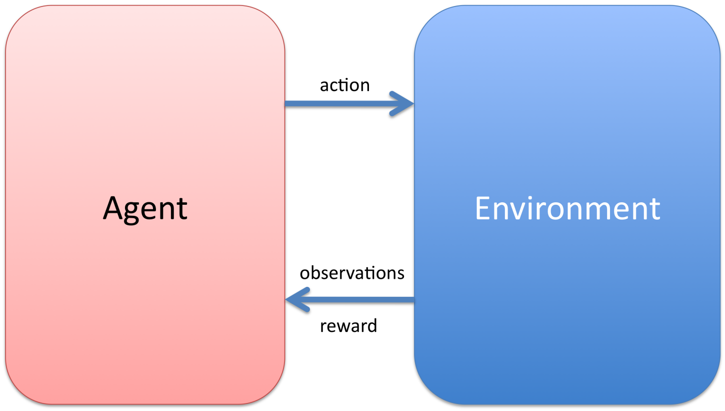

Figure 2.1: The paradigmatic reinforcement learning setting.

In completely observable tasks, the agent directly observes the state of the environment and there is no hidden information. Completely observable reinforcement learning tasks have the Markov property: the current state of the environment is conditionally independent from previous states. Completely observable reinforcement learning tasks are known as Markov Decision Processes (MDPs).

In partially observable tasks, the agent does not directly observe environment state and there is hidden information. In these tasks, the environment state still has the Markov property and is conditionally independent from previous states but is not directly observable by the agent. Instead, the agent must overcome uncertainty as to

[1]An observation can be a single discrete symbol, a vector of discrete symbols, a continuous value, a vector of continuous values, or a vector of both discrete symbols and continuous values. This thesis investigates only tasks with discrete symbols.

the true state of the environment. Partially observable reinforcement learning tasks are known as Partially Observable MDPs (POMDPs). We refer to the hidden state of the environment as latent state as the agent cannot directly observe it.

The partial observability of POMDPs causes environmental states to be perceptually indistinguishable to the agent. For example, if there are two states in an environment S_1 and S_2, and in both states an agent observes observation O_1, then states S_1 and S_2 are said to be *aliased* with one another. In this thesis, agents use working memory models to maintain knowledge of history and use that knowledge to disambiguate perceptually aliased states.

Approaches that solve reinforcement learning problems can be distinguished as *online* and *offline*. In the online case, an agent directly experiences the environment and updates its knowledge of the task after selecting each action. In the offline case, an agent can learn either from direct experience and update its knowledge periodically, it can learn from traces of other agents in the task, or it can even learn by interacting with a model of the task. This thesis considers only the online setting, as we are interested in how agents learn to use memory while they simultaneously learn to perform a task through direct interaction with task and memory.

MDPs can be solved in a variety of ways, including offline dynamic programming, offline or online Monte Carlo sampling algorithms, and online temporal difference algorithms. We are interested in the foundational temporal difference algorithms of Q-learning (Watkins, 1989; Sutton & Barto, 1998) and Sarsa (Rummery & Niranjan, 1994; Sutton & Barto, 1998), and also discuss Monte Carlo for its relevance to eligibility traces. While these algorithms lose their convergence guarantees when applied to POMDPs, they can still be successfully applied in some POMDPs.

Temporal difference methods track the estimated values of state-action pairs[2]. A state-action value, also known as a Q-value, is the expected reward of taking an action

[2]This is not entirely accurate; temporal difference algorithms may also track state values, and not state-action values, but this distinction is irrelevant to the work presented in this thesis.

in the specified state and then following a specific policy from that point forward, where a policy is a mapping of observations to actions.

Temporal difference methods update Q-values by selecting an action, receiving an observation and reward, and then immediately updating the relevant Q-value to reflect the actual reward that the agent received. These Q-values can be calculated by any number of possible functions; in this thesis, our framework uses a look-up table mapping state-action tuples to values.

The primary difference between Q-learning and Sarsa is that Q-learning is an off-policy method and Sarsa is on-policy. Q-learning, like other off-policy algorithms, updates its estimate for the value of the optimal policy while following another policy; if that other policy maintains certain properties then Q-learning is guaranteed converge to the optimal policy in MDPs. Sarsa learns on-policy, which means that the Q-values that it learns estimate the value of the policy that it is currently following, and therefore Sarsa must follow a greedy-derived policy as it learns in order to learn an optimal policy. Like Q-learning, Sarsa is guaranteed to converge to the optimal policy in MDPs when certain properties are maintained.

Monte Carlo algorithms are more general than temporal difference methods in that they can learn from any set of traces in the environment, whether those traces are generated via online interaction or simply provided to the learning algorithm in offline batch mode. Monte Carlo algorithms can also learn Q-values but do not bootstrap their value updates off of existing estimates, instead their updates are derived only from observed returns.

While both temporal difference and Monte Carlo methods can be used to learn optimal policies in MDPs, they actually learn different estimated values (Sutton & Barto, 1998, p. 144). To blend properties of each algorithm (and thus bridge the estimated values during online learning between temporal difference and Monte Carlo algorithms), eligibility traces can be used. Eligibility traces parameterize both Sarsa and Q-learning according to an eligibility trace decay rate (λ), and are then designated as Sarsa(λ) and Q-learning(λ). An eligibility trace decay rate of zero is equivalent to pure temporal difference learning, while a rate of one is equivalent to pure Monte Carlo learning (where Monte Carlo updates minimize mean-squared error of the experienced observations and rewards). As the decay rate slides between zero and one, the relative contribution of temporal difference learning and Monte Carlo learning shifts.

Notationally, Q-learning(0) refers to pure temporal-difference Q-learning; Q-learning(1) is Q-learning with pure Monte Carlo updates; and Q-learning(λ) is Q-learning with a combination of temporal difference and Monte Carlo updating as weighted by the λ parameter.

The practical difference between temporal difference and Monte Carlo approaches is in how reinforcement learning updates propagate through the estimated values. Consider the following situation: an agent is in state s, selects action a, transitions to state $s^{'}$ and receives reward r. A temporal difference method will update the Q-value for the state-action pair (s, a) according to the observed reward and an estimate of the state-action value for best action that could be selected in state $s^{'}$. A Monte Carlo method, however, will update the values for all states that have previously been experienced in an episode according to the reward that was observed. Conceptually, temporal difference methods perform one step backups while Monte Carlo methods back up that reward across all observed transitions.

As noted in Sutton & Barto (1998, p. 191), there is no reliable or useful way to know what the optimal setting of the decay rate is to solve a given MDP and the best setting of λ must be determined on a case by case basis. When using eligibility traces in this thesis, we set λ by performing parameter sweeps and using the best performing value.

Online reinforcement learning in POMDPs is considerably more difficult, although the same methods of temporal difference and Monte Carlo methods can be applied.

Sarsa(λ) has been shown to perform well in POMDPs with good memoryless policies, or where there are good policies that are mappings of observations to actions (Loch & Singh, 1998). In general, using eligibility trace decay rates between 0 and 1 perform best in POMDPs because they do not entirely bootstrap off of estimated values when performing updates (in contrast to pure monte carlo), but they do back up rewards experienced in a single trace across multiple observed transitions (in contrast to pure temporal difference learning) (Sutton & Barto, 1998).

There are two costs to using eligibility traces: both increased memory complexity and time complexity. First, additional information must be maintained in memory and relating to the eligibility of previously experienced state-action pairs. Second, while pure temporal difference methods update a single Q-value after each decision, methods using eligibility traces update values for all previously experienced state-action pairs[3].

2.2 Working Memory

The agents that are explored in this thesis learn to use short-term memories, where the contents of memory are reset between task episodes. We say that these memories are types of working memory, and have functional characteristics similar to working memory in humans. Human working memory stores relatively small amounts of immediate knowledge for short time durations that can be reasoned over by cognitive processes. Working memory stores knowledge that is relevant to a task at hand.

We explore two different types of working memory, as described in Section 3.3.

[3]In practice only state-action pairs with an eligibility above a certain threshold value are updated, thus preventing the number of state-action pairs that must be updated from growing infinitely large

2.3 Actions to Use Memory

An artificial agent exists in the context of a supporting framework that defines how an agent's internal components interact and pass information to and from each other (our agent framework is specified in Chapter III); we refer to this supporting framework as an agent's architecture. Actions that use memory include those that encode and store information to memory, actions that retrieve knowledge from memory and make it available to the agent's immediate cognitive processes, and actions that actively maintain knowledge in memory.

An agent's architecture specifies a procedure by which an agent selects actions. At each decision point an agent is presented with a set of possible actions and selects one; for example, the agent might select a storage action that stores an agent's current perception to memory. We say that actions selected in this manner are *deliberate*.

Other actions are not selected deliberately, but instead are fixed in the agent's architecture. The agent is not presented with a set of possible actions but instead the agent automatically performs an action. For example, knowledge from long-term memory might be retrieved automatically depending on contextual cues from an agent's perception. We say that these actions are *architectural*.

Actions that use memory may either be deliberate or architectural. Deliberate memory actions participate in an agent's decision making process along with other actions, such as available actions in the environment. Architectural actions happen automatically; one possible case could be when information is automatically stored to a long-term memory mechanism.

One important distinction between deliberate and architectural actions is that as an agent accumulates knowledge, it can change which actions it performs, while architectural actions are fixed throughout an agent's lifespan. Thus an agent can learn when to select deliberate actions.

In this thesis, we are concerned with how agents learn to use memory, and we explore agents that use memory via deliberate actions. In certain cases, we modify an agent's architecture to force it to take particular actions in certain situations in order to observe the effect it has on agent task performance. In other prior work (described below), certain operations on memory, such as when to store information to memory, are always architectural. We also discuss the implications for our work and what our results suggest regarding architectural memory actions in Chapter XI.

2.4 Prior Work Learning to Use Memory

Initially, reinforcement learning research explored learning to use memory as an approach to overcoming partial observability. While eligibility traces had demonstrated some success in learning good policies in POMDPs, endowing a reinforcement learning agent with an internal working memory was one possible way of maintaining knowledge of history over time in a task.

Littman (1994) is cited as the first to pursue this direction, and used Sarsa with a single bit of computational working memory in mazes. In addition to the environmental actions available to the agent in the mazes, the agent could toggle the state of working memory between two binary states. Conceptually the agent could be viewed as tying and untying a string around its finger, and incorporating that bit of information into its internal representation of environment state. Results with this approach were mixed: some mazes could be solved using memory and others could not. No conclusions were made characterizing the problems that could or could not be solved with this memory.

Using the same conceptual framework, Peshkin *et al.* (1999) and Meuleau *et al.* (1999) compared Sarsa to VAPS (Baird & Moore, 1999), finding that there were properties of VAPS that made it more suitable to partially observable domains. VAPS is an offline algorithm that finds an optimal policy in a policy-graph.

While the work discussed above involved learning to use a single bit of memory, Lanzi (2000) explored using k bits of internal memory, and more specifically performed a qualitative analysis of the behaviors that agents engage in when they fail to converge to the optimal policy while learning to use memory. Lanzi observed that agents engage in the same sort of looping behaviors endemic to applying online temporal difference algorithms to partially observable settings, and furthermore that it was difficult to characterize tasks in which agents would have difficulty learning optimal policies. Extending this work, Lanzi & Wilson (2000) explored how varying the number of bits in memory affects convergence, and found a tradeoff. Agents endowed with memories with more bits converge to better overall policies, but take more experience to converge while performing worse overall before convergence is reached.

More recently, Todd *et al.* (2008) developed a biologically-inspired model of gated memory. In gated memory, symbols are stored directly from the environment to memory. Their intent was to model human learning in partially observable domains, and they compared results to human data for two problems. Their model uses an actor-critic reinforcement learning algorithm, where there is an independent actor for each slot of working memory along with an independent actor for the environment. Each actor selects an action on every time step; for the gating actors, this action is either to open the gate and update the contents of that memory slot or to keep the gate closed and preserve the contents of the respective slot. While a model was proposed and demonstrated to learn in some small POMDPs, the authors note that the work was a first step towards a more comprehensive model and significant work remained to be done in order to extend their approach to more complex POMDPs.

Zilli & Hasselmo (2008a, 2008b) focused on very simple grid-world tasks inspired by the rat mazes common to experimental psychology literature. They considered a gated memory mechanism that could store a single perceptual observation to memory, as well as an episodic memory mechanism that could perform a cue-based retrieval using the observation from the current location, and then retrieve memories that are temporally adjacent to the previously retrieved memory in order to obtain the next observation symbol that was perceived. They took an analytical approach and developed algorithms that could determine whether a specific memory mechanism could support optimal behavior in a specific task. They further extended this work to be able to determine whether the combination of gated working memory and episodic-like memory could support optimal behavior. Note that this work did not address online learning, just whether the optimal policy in the task could be represented in conjunction with a specific computational memory model.

Some prior work involved partial characterizations of situations in which it was difficult to learn to use memory, and described the learned behaviors that agents exhibited when they failed to effectively learn to use memory. Lanzi (2000) discusses a chicken-and-egg problem of learning to use memory while simultaneously acting in the environment. Lanzi suggests that exploring the space of environment actions before exploring the space of internal actions can be helpful to achieve convergence, and thus having independent exploration rates for internal and external actions may help speed convergence and increase the number of correct solutions that agents converge to. We also encounter chicken-and-egg problems involving learning to use memory and discuss their significance in Chapter IV. Rather than explore different parameter settings for memory and environmental actions, we explore the effects of making some memory actions architectural.

Arai & Sycara (2001) also noted that when agents fail to learn to use memory, they fail by exhibiting looping behaviors (as noticed by Lanzi, above). Arai & Sycara used an episodic-like memory mechanism to store transitions in episodic tasks, and only reinforce each unique transition once in order to dissuade an agent from looping over transitions repeatedly. For the settings that they explored (grid-world mazes), their algorithm compared favorably to Sarsa. Note that control of the episodic-like memory mechanism was fixed in their architecture and that their agents did not learn to use memory, but learned to behave in the environment while being provided with knowledge from memory as part of the agent's representation of state.

Zilli & Hasselmo (2008c) investigated learning to use gated working memory and episodic-like memory in a series of rat maze tasks abstracted as grid-world tasks. They investigated the performances of agents that learned to use each memory mechanism independently as well as agents that learned to use both memories simultaneously. For some of the tasks, the optimal solution could be represented by an agent endowed with gated working memory, but those agents were unable to learn the optimal solution. However, agents endowed with episodic memory could learn to perform the task optimally. They concluded that as the temporal length of the task increased, agents had more difficulty relying on working memory and had to rely on episodic memory instead. In this thesis we also explore tasks with extended temporal lengths, but find that agents endowed with a specific type of working memory can solve these tasks.

2.5 Learning to Use Episodic Memory

In prior work, we investigated agents that learned to use two types of memory: episodic memory and bit memory (Gorski & Laird, 2011, 2009) within the context of the Soar cognitive architecture (Laird, 2008). This prior work motivated how we will explore the space of memory models. In it, we demonstrated that an agent could learn to use episodic memory in three different ways. However, agents could not effectively learn to use bit memory, highlighting differences between the memory mechanisms and suggesting a path for future research.

Our experiments were performed in a simple discrete domain called *Well World*. An agent in Well World has two important internal states: either the agent is thirsty and is rewarded for quenching thirst by drinking water from a well, or it is not thirsty and is rewarded for remaining safe at a special location in the domain. The complex dynamics arise because the domain is partially observable, and in order to act well in the domain the agent must maintain some memory of which well it last drank from in order to consume water from an alternate well the next time it is thirsty.

In this domain, agents learned to use Soar's episodic memory in three different ways. The first two ways of using episodic memory involved learning to perform specific cognitive capabilities: the first was to support virtual sensing, while the second was to support remembering past actions. The third way of using episodic memory was a surprising result: agents learned to use episodic memory as a simple bit memory.

In the first experiment, agents were provided with multiple cues, only one of which would deterministically result in the correct episodic memory being retrieved. Agents had to learn how to act in the environment before becoming thirsty, to select the correct retrieval cue after becoming thirsty, and then to associate a sequence of actions in the environment with the information contained in the retrieved episodic memory. We demonstrated that agents could successfully learn to use episodic memory to support optimal behavior in this task, and in doing so, learn to perform virtual sensing.

In the second experiment, agents were provided with individual features that could be used to retrieve episodic memories, and by performing successive retrievals agents could learn to construct an episodic memory retrieval cue. In order to reliably retrieve the episodic memory that would correctly inform which well to visit when thirsty, an agent had to learn to perform a retrieval from memory using two features and not using the irrelevant features. We demonstrated that agents could learn to use episodic memory in this task, and learn to perform the task optimally while doing so.

In the third experiment, the task was structured to elicit the learning of the cognitive capability to remember past actions. Learning to remember past actions, for this particular domain, was more complicated than we expected it would be. Learning past actions in this task involved an agent, upon becoming thirsty, remembering the last time it was thirsty and then advancing episodic memory to remember the action that it last performed when thirsty, and then condition its current task behavior upon that information.

It was difficult for an agent to learn this behavior, however. When the agent was unable to learn to remember past actions, it instead used episodic memory as an effective bit memory. To do so, the agent used the retrieval buffer as a bit that it could deliberately set by making a retrieval. When the retrieval buffer was empty, the agent always went to a particular well when it became thirsty. If that well contained water, it drank; if not, it performed an episodic memory retrieval and then used the knowledge that a retrieval had been performed (and not any details of which knowledge had been retrieved) to move to the other well. This behavior was not the optimal behavior in the domain, but was stable enough that agents that learned this behavior never ended up learning to perform the optimal behavior.

After observing this learned behavior, we experimented with a reinforcement learning agent that was endowed with a simple bit of memory rather than an episodic memory. It is clear that a single bit of information is sufficient to perform optimally in the domain, as the agent must only know which well it last drank from in order to visit the alternate well the next time it becomes thirsty; furthermore, agents were able to learn to exhibit the optimal task behavior when using episodic memory as an effective bit memory.

Figure 2.2 shows the performances of agents that learned to perform in the Well World task (Gorski & Laird, 2011). In this plot, the optimal behavior is zero reward per action. When an agent was provided with fixed control knowledge as to how and when to toggle its internal memory bit, the agent converged to the optimal behavior within five hundred actions (labeled "Fixed strategy"). When the agent drank from one well, the agent always set the contents of its memory to one of two values; when the agent drank from the other well, the agent always set the contents of its memory to the other of the two values. The architectural knowledge of when to set memory and which values to set them to were the extent of the architectural knowledge that agents were provided with. This experiment demonstrates that a single bit of memory is indeed sufficient to support both acting optimally in the environment, as well as learning to act optimally in the environment. Although agents were provided with background knowledge as to how to control memory, they were not provided with any knowledge as to how to act in the environment given knowledge that was in memory.

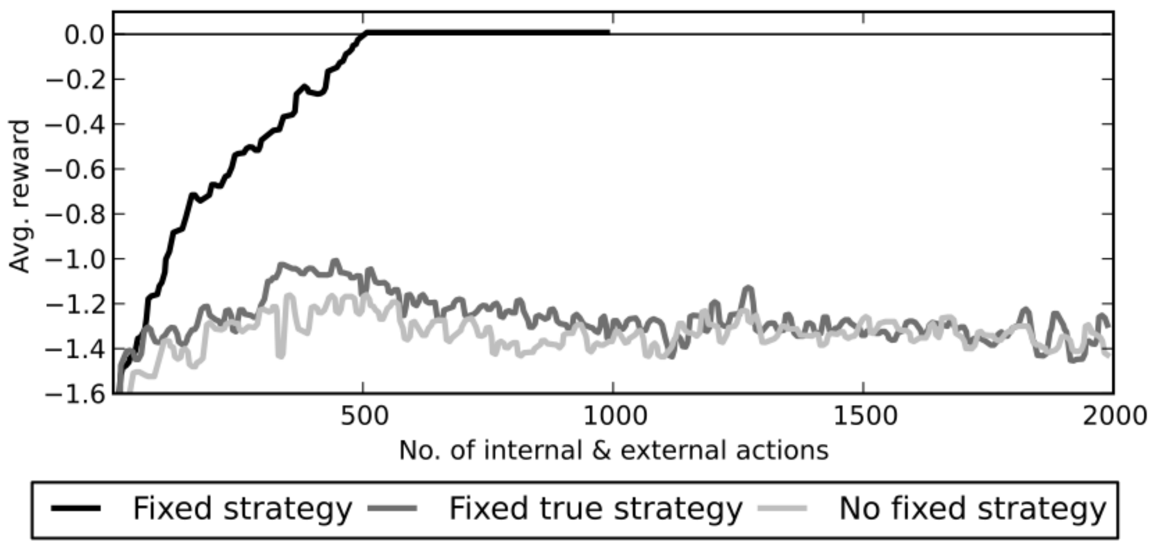

Figure 2.2: Performances of agents learning to use bit memory in Well World for various amounts of a priori procedural knowledge.

Although the agent that was provided with control knowledge learned to act in its environment optimally and used memory to support this optimal behavior, it did not learn to select actions over memory. When learning to use memory, an agent must learn two distinct types of knowledge: it must learn which actions to select to control knowledge in memory, and it must also learn which actions to select in its environment given the contents of memory. Effectively, an agent must learn what to put into memory and then, once knowledge is present in memory, how to act based on that knowledge. The agent that was provided with control knowledge learned only the latter, as the former was provided to the agent as architectural knowledge.

An agent that wasn't provided with any architectural background knowledge as to how to control memory in the environment was unable to learn to use memory ("No fixed strategy"). In order to perform optimally in the task, the agent had to learn to use memory in both senses: it had to learn when to select memory actions in order to store perceptual information to memory, and it also had to learn how to act in the task given the knowledge that was stored in memory. Although the memory was sufficient to support optimal behavior in the task (as exemplified by the agent discussed above), agents were unable to learn to use memory effectively in this task.

We also examined the behavior of a third agent condition that was provided with partial background knowledge as to how to control the contents of memory. This agent had architectural knowledge that dictated it set the bit to a particular value when drinking from a particular well. Our intent was to provide sufficient knowledge to bootstrap the agent's learning behavior so that it could learn to set memory contents to the opposite value when drinking from the other well, as well as how to act in the task based on the contents of memory. This agent bridged the differences between the first agent (when the agent was provided with complete architectural background knowledge as to how to control memory) and the second agent (when the agent was provided with no architectural background knowledge as to how to control memory).

In this condition, the agent was unable to learn to perform the task, and performed very similarly to the agent that was provided with no architectural background knowledge. Although partial background knowledge was provided, that agent was unable to learn the remaining control knowledge, and thus was unable to learn to use memory.

The differences in agents that learned to use Soar's episodic memory and bit memory are large, especially considering that agents were able to learn to use Soar's episodic memory as a bit memory. These observed differences in behavior are what motivated our approach to exploring the space of memory mechanisms. By identifying two memory models that when applied to the same task result in very qualitatively different levels of performance, we can further investigate how those memory models are different and then explore which differences are most responsible for the differences in behavior.

In the Well World example, we identified several important ways in which the two memory models differed (Gorski & Laird, 2011):

- Episodic memory has unlimited capacity, whereas bit memory has a very limited capacity. Alternatively, knowledge in episodic memory has unlimited persistence, whereas knowledge in bit memory persists only until memory is toggled.
- Knowledge is stored to episodic memory architecturally and automatically.
- Knowledge in episodic memory is a comprehensive snapshot of a situation, while knowledge in bit memory is abstract.
- Knowledge in episodic memory can be retrieved via cue-based partial matches. By contrasting the two memory models in the context of the task in which differences in behavior and performance were observed, we are able to identify paths for future research. We will modify the memory models along these relevant dimensions, and determine which of the characteristics are associated with which task characteristics.

CHAPTER III

A Framework to Explore the Dynamics of Memory and Task

In order to explore how a reinforcement learning agent can learn to use memory to perform a task, we require a framework specifying the interfaces of memory, reinforcement learning, and task. We specify this framework conceptually in this chapter in sufficient detail such that our implemented software framework could be duplicated.

This chapter begins by proposing a conceptual framework to support an agent that learns to use memory (Section 3.1), and continues by discussing the various dimensions along which we explore reinforcement learning (Section 3.2), memory models (Section 3.3), and the TMaze task (Sections 3.4 and 3.5).

3.1 A Conceptual Framework for Learning to Use Memory

In order to explore how a reinforcement learning agent can learn to use memory to perform a task, we created a framework in which a reinforcement learning agent is endowed with an internal memory mechanism. Figure 3.1 shows the conventional reinforcement learning paradigm, where an agent selects actions in the environment and in return the environment provides the agent with an observation and a reward (Sutton & Barto, 1998). Figure 3.2 shows the conceptual framework that we developed in order to explore how an agent learns to use memory, and which we used to support our computational experiments.

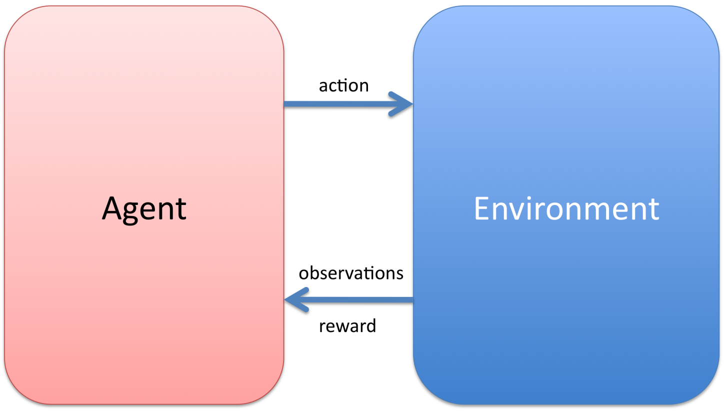

Figure 3.1: The paradigmatic reinforcement learning setting.

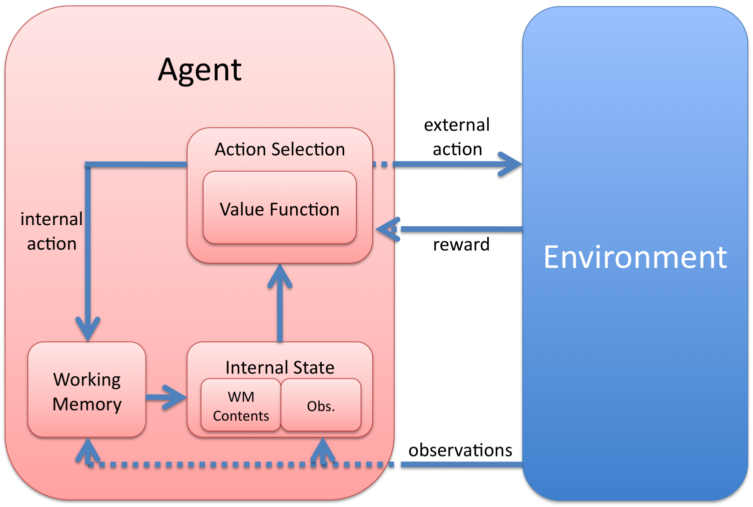

Figure 3.2: A conceptual framework endowing a reinforcement learning agent with internal memory.

As in the conventional paradigm, the agent selects actions in the environment and in return receives an observation and reward. To better illustrate the inner workings of the reinforcement learning components, we show working memory, internal state and action selection components as separate modules. In addition to selecting actions in the environment, the agent also selects actions that control its internal memory mechanism.

In the conventional reinforcement learning paradigm, the observation from the environment is used to represent the state of the environment. In completely observable MDPs, this observation of the environment is exactly the environment state. However, in POMDPs, this observation is a function of latent state, as described in Section 2.1.

By contrast, our extended framework concatenates the observation from the environment with information from memory and this composition serves as an agent's representation of task state. We refer to this representation as an agent's *internal state*. This internal state is then used as the state component of (state, action) pairs in the tabular value function.

Environment actions in reinforcement learning tasks are specific to the task provided to the agent, and are provided by the environment to the agent. In our framework, where agents are endowed with internal memory mechanisms, actions that control memory are provided to agents in a similar manner. Agents are provided with different actions to control memory depending on the specific memory mechanism that is available to be controlled. The actions that are available to an agent to control memory depend on the memory mechanism that is being controlled and the current contents of that memory mechanism; we discuss the space of actions available over memory in more depth in Section 3.3.

In our framework, a single action is selected at each decision point: the agent selects either an internal or an external action at each decision, rather than a combination that occurs simultaneously. Peshkin *et al.* (1999) referred to this design decision as *augmenting* the action space. The alternate design decision, *composing* the action space such that every environment action is composed with all possible internal memory actions, is undesirable for two reasons. First, our experimental framework emphasizes simplicity, and augmenting the agent's action space was the most pragmatic design decision. Second, the complexity of the state-action space is higher when actions are composed. Although augmenting the action space could potentially introduce a bias to the learning problem, this can be avoided by not discounting internal actions Peshkin *et al.* (1999), an approach implemented in our framework.[1]

Only the environment provides reward to an agent. For each non-terminal action taken by the agent, it receives a small negative reward, and this reward is the same regardless of whether the action is internal or external. The reward function in the framework never depends on the state of internal memory, it only depends on task state. Although an agent selects only a single action on each time step, if an internal action is selected then the task state does not change and it is treated as a no-op action incurring small negative reward.

An agent's interaction with its internal memory mechanism is analogous to its interactions with an external environment. In the case of the environment, an agent selects actions that change environment state and in return receives an observation that is a function of environment state. In the case of internal memory, an agent selects actions that change the contents of memory and in return receives an observation that is a function of memory state. In fact, some past literature has referred to memory used in conjunction with a reinforcement learning agent as *external* memory, rather than internal (Peshkin *et al.* , 1999); the reasoning being that the memory mechanism is external to the reinforcement learning mechanism of the agent. We, on the other hand, refer to it as *internal* memory because it is internal to the agent and not a part of the external environment; the dynamics of the environment depend only on actions that the agent selects (or does not select) in the environment.

Although the observations provided by memory and environment are provided by independent mechanisms, in our framework an agent's observation is always the union

[1]In fact, all of the tasks that we explore in this thesis have no discount rate as they are episodic in nature.

of the two. From the perspective of the reinforcement learning algorithms that are implemented in our framework, an observation is a single vector of discrete symbols; no distinction is made between symbols provided by the environment versus those provided by memory and no semantics are attached to the origins of symbols.

Our framework is partly inspired by the frameworks used by work described in Chapter II. However, our framework does differ in important ways. In the architecture used by Littman (1994), actions were composed rather than augmented. Peshkin *et al.* (1999) used an offline model-based reinforcement learning algorithm, VAPS (Baird & Moore, 1999), whereas our approach is online and model-free.

In this general framework, we are able to explore the interactions of memory and task by selecting a particular memory model and a particular task instance and observing the dynamics. The following sections describe the specifics of the memory models and tasks that we used in our experiments.

3.2 Reinforcement Learning in Our Framework

As discussed in Chapter II, reinforcement learning is a broad and encompassing term. Our experiments are concerned with a specific reinforcement learning setting, and beyond that our framework makes many commitments in terms of both numerical parameters and architectural policies.

One motivation for our work is to understand how a long-lived agent might learn to use its own internal memory mechanisms over time. On the other hand, we are not seeking to improve learning performance over existing algorithms or techniques. One characterization of our setting is that we are (necessarily) interested in environments that are non-Markovian[2]. Although there are a variety of approaches for tackling non-Markovian problems, we're interested in the online behavior exhibited by agents

[2]In Markovian environments, the utility of memory is obviated as the environment state is directly observable.

that must directly interact with their environment.

Our framework exclusively uses the foundational online temporal difference learning algorithm of Q-learning (Watkins, 1989; Peng & Williams, 1996; Sutton & Barto, 1998). We compared the performance of Q-learning to another foundational online temporal difference learning algorithm, Sarsa(λ) (Sutton & Barto, 1998), and found that Q-learning converged to optimal more quickly across the tasks and memory combinations that we explore. Using Q-learning allows us to model an agent that uses memory while interacting online with an environment, but without requiring a model of the environment dynamics.

Agents in our framework moderate exploration and exploitation with a softmax procedure sampling from a Boltzmann distribution. In experiments using eligibility traces, we use Peng's Q(λ) (Peng & Williams, 1996; Sutton & Barto, 1998) or Sarsa(λ).

Within our framework, the reinforcement learning mechanism does not generalize and lacks any capability for state abstraction. The agent's value function, updated by the reinforcement learning mechanism as it gathers experience in the environment, is represented as a simple lookup table, providing a mapping from internal state to estimated value.

Our framework assumes certain representations about the environment and memory. First, it assumes that the representation of environment and memory state is discrete; there is no support for any continuous representation that does not natively couple with a tabular value function. Actions are also assumed to be discrete, and there is no support for continuous action spaces.

3.3 Memory Models

Our investigation focuses on two memory models: bit memory and gated memory. Both memory models that we used are short-term working memory models: no long-term knowledge persists between task episodes, as memory contents are re-initialized at the completion of each task instance.

In the parlance of this dissertation, *knowledge* will refer to information that is stored in memory. When a symbol enters memory, it is knowledge; when a symbol is provided from the environment to the agent, it is *perceived* as part of an agent's *observation.*

In both models, knowledge in memory is passively maintained. Internal actions available to the agent can change the state of memory; conceptually, this is akin to storing new knowledge to memory and overwriting the prior contents of the model. Other working memory models require that internal maintenance actions be actively selected; due to our framework's commitments to only one action being selected at a time, internal or external, passive maintenance is a necessary abstraction in our framework. Furthermore neither model provides retrieval actions to an agent, and instead the contents of each working memory model always feed directly to an agent's observations.

Although knowledge in memory may be either abstract (bit memory) or concrete (gated memory), in neither case does the agent have any *a priori* semantics as to the significance or meaning of symbols. As in the conventional reinforcement learning paradigm, the agent's goal is to find the best policy; in our framework, policy is represented as a mapping of internal state to action.

3.3.1 Bit Memory

The motivating idea behind the bit memory model is to create the simplest conceivable memory model. In the base case, a bit memory maintains a single unit of knowledge. This unit of knowledge may take on one of two possible values; being binary, we refer to it as a single bit. As a convention, we will use the symbols '1' and '0' to refer to the possible contents of memory.

In the base case, bit memory has a single action with which an agent can control its contents: *toggle*. The toggle action switches memory between the two possible binary values. If bit memory contains the symbol '1', then the toggle action will set the contents of memory to '0', and vice versa. This behavior is illustrated in Figure 3.3.

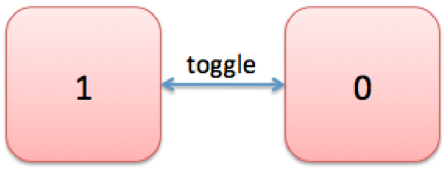

Figure 3.3: Possible states of bit memory and how the toggle action transitions between them.

An agent endowed with bit memory has no *a priori* semantics associating the contents of memory with either actions in the environment, actions over bit memory, or with observations from the environment. Instead, in addition to the standard reinforcement learning problem of learning a policy in the environment (and in our setting, over internal memory), the agent must also learn to associate a particular symbol in bit memory with the underlying environmental state (which is not always directly observable). This aspect of bit memory has subtle implications; we will contrast bit memory with gated memory in Sections 3.3.2 and 3.3.3, as will refer to it in the remaining chapters.

The inspiration for bit memory, as discussed by Littman (1994), is the ability to remember to do a single thing in the future. Littman compares this ability to a person tying and untying a string around their finger in order to remember to do something. Similarly, one bit of internal memory allows an agent to disambiguate between two aliased situations, or environmental states.

3.3.1.1 Dimensions of Bit Memory

The base bit memory model can be modified along a number of dimensions:

- Bit memory can be controlled with actions that set memory contents to a specific value, rather than an action that toggles between possible values. For the base bit memory, this consists of two available actions: one to set memory contents to 0, and one to set contents to 1.

- Bit memory can store more than binary values. It can store ternary values, 4-ary values, and so on.[3] Toggle actions for n-ary bit memory would involve exposing a single toggle action that iterates through n possible values one by one, wrapping around from $n-1$ to 0 and incrementing the contents otherwise. Set actions for n-ary bit memory would simply be extended from two set actions to n, one for each possible value.

- Bit memory can contain more than one slot of knowledge, where each slot consists of distinct units of knowledge that can be individually toggled or set. This is illustrated in Figure 3.4; in (a), there is a single unit of knowledge that can take on one of two binary values, while in (b) there are n slots that can each maintain a binary value. While a three slot memory storing binary values can store the same as an 8-ary memory with a single value, there are additional constraints imposed by the transitions between possible memory states and the actions that are available to change memory contents. Actions over memory toggle or set individual slot contents, rather than manipulating all contents simultaneously.

- Bit memory must have an architectural policy dictating how memory contents are initialized. Memory contents can be initialized randomly, to a fixed value,

[3]Bit memory would then become a misnomer, and instead *abstract memory* or *ungrounded memory* might be more appropriate. For consistency, we will refer to modified bit memory models as bit memory with more than two possible memory contents.

or to a special null value which memory can only be set to but cannot be reset to after an internal action has been selected on that memory slot. In the case of toggle and null-initialized bit memory, the first toggle action changes memory contents to 0 and then subsequent toggle actions behave as in the base case after that. Unless otherwise noted, we initialized bit memory to one of two binary values such that the contents are set to each possible value with equal probability.[4]

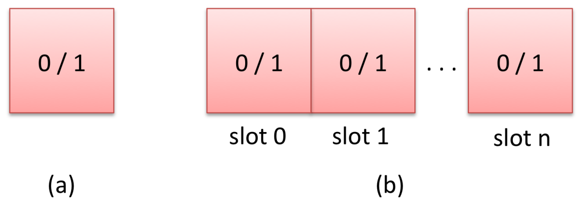

Figure 3.4: (a) Bit memory consisting of a single bit of knowledge. (b) Bit memory consisting of more than one bit of knowledge, where each conceptual memory slot stores a discrete bit.

- The agent framework can be modified to make certain internal actions architectural and fixed, rather than deliberate and learnable. For example, in a certain environment state bit memory is set to a particular value according to a fixed architectural policy. Similarly, the agent framework can be modified so as to prevent the selection of internal actions in specific environmental states.

We explore modifications to the base bit memory model along these dimensions throughout Chapter IV.

3.3.2 Gated Memory

Our gated memory model maintains an observation received from the environment. The *gate* in gated memory is a conceptual barrier between the perceptual

[4]We selected this as the default initialization policy because, after some exploration in the space of possible strategies, we found that this particular dimension of bit memory had little impact on overall learning performance, and it seemed like a reasonable default choice.

buffer and the contents of working memory. When the gate is raised, then the contents of working memory are preserved and perception does not interfere with the maintained knowledge. When the gate is lowered by the selection of the internal gate action available to the agent, the observed perceptual symbol from the environment replaces the current contents of working memory; the gate then is immediately (and architecturally) raised and knowledge in memory is maintained until the gate is again deliberately lowered at some point in the indefinite future. The gate action is the only internal action available to modify the contents of gated memory.

In the base case of the gated memory model, memory is initialized as empty, which is represented by a unique *null* symbol. The null symbol never appears in the environment, and it is the only symbol that can be present in the contents of gated memory that is not directly perceived as an environmental observation.

As mentioned in the previous section, knowledge in bit memory is abstract and any association between memory contents and environmental observations must be learned. In contrast, knowledge in gated memory is grounded in perception because a symbol can only enter memory when it is directly observed. An agent cannot use internal actions to change the contents of gated memory to an arbitrary value as it can with bit memory. While an agent must still learn to use gated memory while learning to act in the environment, it does not face the additional learning problem of settling on a policy for how to associate storing knowledge in memory to environmental observations as that association is provided implicitly.

This computational model of gated memory is inspired both by our understanding of human working memory (Baddeley, 1986; Gluck *et al.* , 2007), as well as by prior models of computational working memories used in conjunction with reinforcement learning (O'Reilly & Frank, 2006; Zilli & Hasselmo, 2008c). Although working memory traditionally refers to both the ability to maintain a representation as well as mentally manipulate it (Baddeley, 1986; Zilli & Hasselmo, 2008c), we are concerned with only the former aspect in this work.

3.3.2.1 Dimensions of Gated Memory

The base gated memory model can be modified along two dimensions:

- Gated memory can contain more than a single slot of knowledge, as bit memory can. If there are n slots of knowledge in gated memory, then n unique internal gate actions are available to allow an observed symbol to be stored in each corresponding slot.

- The agent framework can be modified so as to enforce specific, architectural behaviors over internal actions. This means that agents would select or be prohibited from selecting internal actions in certain internal states. For example, an agent might be required to gate memory when observing a specific symbol in the environment and it has not already been gated to memory. This was also a dimension of bit memory, above.

3.3.3 Brief Discussion of Abstract and Concrete Knowledge

As mentioned previously in this section, bit memory contains abstract knowledge (arbitrary symbols) while gated memory stores concrete knowledge (perceptually-grounded symbols that have been observed). This will prove to be an important differentiation between the two models in Chapter IV, and there are two contrasts that we highlight here in order to foreshadow later empirical results.

First, gated memory allows for a symbol to be stored to memory only when it is being observed. If, in the course of a task episode an agent observes only a proper subset of all observations that an environment could have provided, then the agent is implicitly restricted in the space of possible memory states that it could have been in throughout the course of the episode, and this in turn restricts the space of possible internal states that the agent could have been in during the episode.

In a memory model that contains abstract knowledge, however, all internal memory states are reachable during the course of an episode, and there is no implicit correlation between task instance and possible internal memory states.

The second contrast follows from the first. During the course of an episode, if the agent does not observe a particular symbol, then it cannot store that symbol to gated memory. This prevents the agent from unintentionally tricking or confusing itself. However, with bit memory, an agent can switch mental states, even if the long-term effects of this mental switch are unfavorable for the agent's performances. As the agent has no representation of history outside of the contents of memory, the effects of this mental trickery can prevent the agent from learning to perform in the environment.

3.4 TMaze

The TMaze domain is a collection of tasks. It includes a base TMaze task, as well as tasks that extend the base task with parameterizations along specific dimensions. This section discusses the motivations for using the TMaze task, describes the task and its dynamics. We use the base TMaze as well as its parameterizations throughout the empirical work described in this thesis.

The base TMaze task is illustrated in Figure 3.5. There are two locations in the task: the starting location, identified as A/B; and a location identified as C where the agent must choose between two terminal environmental actions, where one terminal action leads to positive reward and the other negative reward. In the first location, an agent observes one of two symbols, either A or B, which correspond to the identity of the positively rewarding terminal action. An agent has only a single action available to it in the environment, which transitions it to the second location. In this location, an agent can only observe C; in order to recover environment history about which symbol was observed in the first location, it must rely on internal memory. In the second location, it must select from between two actions, one of which will result in positive reward if an agent observed A in the first location, and the other action which will result in positive reward if an agent observed B in the first location. If an agent selects the wrong action, it receives negative reward.

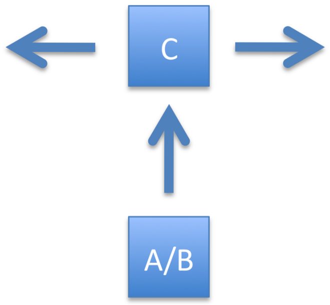

Figure 3.5: The base TMaze task.

We use the TMaze domain to support our experiments because the base TMaze task is the simplest task that can be formulated that requires information about task history. We use the simplest possible tasks because it allows us to isolate the effects of using memory to preserve salient information of task history. If instead we explored learning to use memory in more complex tasks, then it would be difficult to analyze the resulting agent behaviors to determine which task characteristics complicated learning and why.

We call our tasks TMazes because of their similarity to a class of tasks used extensively throughout animal psychology literature to study learning and memory (Tolman & Honzik, 1930; Malone, 1991).

TMaze is an episodic task, but the correlation between symbol in the first location (the *salient symbol*) and the action that leads to positive reward in the second location (the *choice location*) persists across episodes. An agent that can maintain knowledge pertaining to a salient symbol can learn an association between that knowledge and the correct terminal action. Conceptually, the salient symbol that the agent observes in the first location is a sign, telling the agent which direction it should later move;

it is the task of the agent to learn both to associate knowledge in memory with the sign symbol, and to condition a later action upon that knowledge.

In the TMaze domain, an agent receives $+1.0$ reward for taking the correct action in the choice location as designated by the salient knowledge in that episode, and -1.0 for selecting the wrong terminal action. For every other action that the agent selects, either an external action or an internal action, the agent receives -0.1 reward.

As mentioned previously, observations and actions in TMaze are free of any *a priori* semantics; the agent must learn through direct experience with the environment the effects of actions and the correlation between salient knowledge and which actions are positively rewarding.

All actions in TMaze are deterministic, and TMaze observations have discrete representations.

In order to successfully perform the base TMaze task, an agent must:

1. Encode and store knowledge of the symbol presented in the first location, either A or B.

2. Move to the choice location.

3. Not modify the contents of internal working memory before the next step (i.e. not gate memory or toggle bit memory).

4. Associate the stored knowledge with available actions and select the correct action in the environment.

3.5 Task Dimensions and Parameterized TMazes

The base TMaze task suits our purposes because of its simplicity, but it also has the advantage that it is easily extensible. We extend the base TMaze along dimensions that relate to how salient symbols must be maintained in memory and used to inform decisions in the task; these dimensions of task are directly related to how an agent must learn to use memory.

The five dimensions of task that we explore are:

1. Temporal delay: the temporal distance between when a salient observation is made and when an environment action depends on knowledge of it, measured both as the number of actions and number of discrete states between the salient observation and an environmental action that depends on knowledge of the salient observation.

2. Number of dependent actions: the number of distinct environment actions that depend on knowledge of the salient observation.

3. Number of distinct symbols: the number of distinct salient observations that can be made in the starting location.

4. Concurrent knowledge: the number of different salient observations of which knowledge must be maintained in memory in order to act correctly in the task.

5. Second-order knowledge: knowledge about the relationship between a salient observation and an observation made in another location (for example, equality and inequality relationships).

Chapters V - XI of this thesis correspond to a dimension of task (as listed above), as well as the base TMaze task. In each chapter we: describe what the dimension of task is, detail how the TMaze task is parameterized along that dimension of task, and present results of our experiments in those parameterizations.

CHAPTER IV

Experimental and Analytical Methodologies

Before presenting our experiments, we discuss our experimental methodology in Section 4.1 and introduce the analytical techniques that we use in Section 4.2. Although the analyses for each set of experiments varies, we apply certain techniques repeatedly and we introduce them first so that the reader may more clearly understand the results of the analyses in each experimental section.

4.1 Methodology for Evaluating Learning Performances

We empirically evaluate how agents can learn to use memory with a two phase methodology. The first phase involves observing the performance of the two working memory models in the task or set of related tasks. For example, when exploring TMazes parameterized along the temporal delay dimension in Section 6.1, we explore several different versions of the temporally extended TMaze, and not just a single version of the task. Through the initial exploration of a task we identify differences in performance between memory models that relate to the specific task characteristic.

Using the measurement and analysis techniques that we discuss in Section 4.2, we examine behaviors that individual agents exhibit and explore state diagrams that allow us to identify trends across task and memory configurations. Performing this analysis allows us to contrast the capabilities afforded to agents using different memory models and to make hypotheses about specific aspects of a memory model, grounded in a specific task, that are responsible for good or bad performance.

In the second phase of our research, we modify memory models in order to change these specific characteristics, empirically testing our hypotheses to determine which characteristics of memory are relevant to performance along which task dimensions.

For example, in Section 5.2 we make modifications to the bit memory model in order to better understand which specific aspects of bit memory are responsible for an agent's failure to learn to perform the base TMaze task. We modify bit memory along dimensions that we identified as important in our analysis from the first phase, and test predictions about which differences between bit and gated memories are responsible for differences in performances.

4.2 Analysis Techniques

Many of the analysis techniques that we use show up repeatedly in the course of our experiments. In this section we briefly describe them so that the presentation of experiments and results isn't bogged down by the additional explication of the analysis as well.

In every case, the first and most straightforward analysis that we use is to plot the mean learning curves for each condition in an experiment, typically for between one hundred and one thousand agent trials (or subjects). We use the mean learning curves to perform both objective and subjective analyses. We use them objectively by visualizing the average performance and comparing it to the optimal performance possible; visualizing whether a mean performance converges to optimal or not tells us whether a task is tractable for a given experimental condition. Additionally, we use them for subjective evaluations by comparing the mean performances of one experimental condition to others, and qualitatively characterizing the relative performances. For example, the mean learning curve for one condition may converge to a higher average reward per episode than that of another, which would set the stage for additional analysis to explain the observed differences in performances.

4.2.1 Individual Behavior Analysis

In order to explain observed mean performances, we perform individual behavior analyses. This involves looking at a set number of agent traces (typically twenty), observing the internal states that they were in and the actions that they selected. By observing the behaviors that an individual agent exhibits across consecutive episodes, we can characterize aspects of the behavior that are of interest. Typically we are interested in the stable, convergent behaviors of agents and thus perform the analysis well after the mean learning curves have converged.

An example of the individual analysis that we perform is found in the base TMaze experiments discussed in Section V. There, we characterize behaviors that agents exhibit over their internal bit memory while in the choice location. This analysis helps us understand why agents were unable to learn the optimal behavior in the task, and by identifying the behaviors that agents exhibit after learning stable behaviors we can better describe why the particular task and memory combinations that we explore do not exhibit optimal behaviors.

In some cases, agents exhibit multi-modal behaviors. When performing individual behavior analysis, we group agents exhibiting similar behaviors and qualitatively characterize them. These data are then presented in table form, listing the percentages of agents exhibiting each type of behavior.

4.2.2 State Diagram Analysis

Both visualizing mean learning curves and examining and characterizing individual behaviors are methods for analyzing the empirical performances exhibited by agents in the course of learning. After performing the objective and subjective empirical analyses as above, we then analyze the state diagrams of an agent's task and memory configuration in order to understand the specific aspects of the learning problem that make learning to use memory more or less tractable.

A state diagram is a directed graph, where states are nodes and actions are edges. Each state in the graph is a unique state configuration in the underlying system; in the TMaze setting, each state is a tuple consisting of three features: the salient symbol that is observed, the current observation provided to the agent in its current location, and the current contents of an agent's memory. Each action, then, transitions from one state to another.

Figure 4.1 shows the state diagram for a completely observable TMaze, for an agent lacking any sort of memory. In the completely observable TMaze, an agent observes two features from the environment: the observation that an agent would normally receive in the standard TMaze (that is, salient knowledge in the starting location and then C in the choice location), but it also observes a second feature which is always the salient knowledge.

The agent begins in one of two initial locations, depending on which version of the task is being used for the current episode. In the starting locations, the only action available to the agent is "up", which transitions the agent to the choice location. In the choice location, if the agent selects the correct action it receives positive reward (a smiley face), if the agent chooses the wrong action it receives negative reward (a frowny face).

We present the state diagram for the completely observable TMaze in order to contrast it with the state diagram for the standard, partially observable base TMaze task for an agent lacking memory. In this setting, an agent cannot learn to perform the task, and cannot achieve a mean performance of better than guessing as to the correct action in the choice location (as it has no knowledge of the salient symbol in memory, and cannot perceive it). The state diagram for this setting is shown in

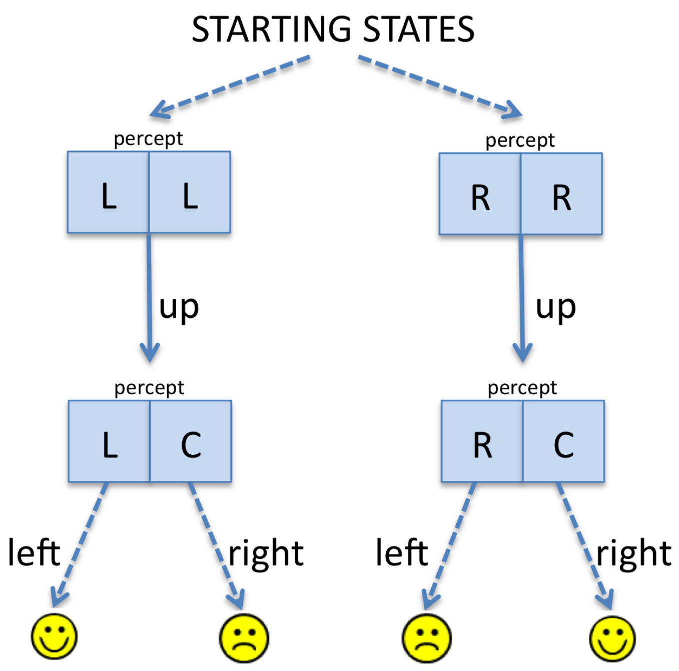

Figure 4.1: State diagram of a completely observable TMaze task.

Figure 4.2.

In this diagram, there is a hashed rectangle drawn around the states LC and RC. This indicates that the two system states are aliased from the perspective of the agent, and that the agent cannot distinguish between the two states using its available internal state information. Although the agent in this configuration lacks internal memory, even agents endowed with internal memory can have aliased states.

4.3 Parameter Selection Methodology

We briefly address practical reinforcement learning methodology issues: our process for selecting parameters for our experiments, our process for making algorithm and architectural design decisions, and experimental design in terms of the number of subjects and amount of experience measured.

In our reinforcement learning framework, there are a number of free parameters:

- Learning rate

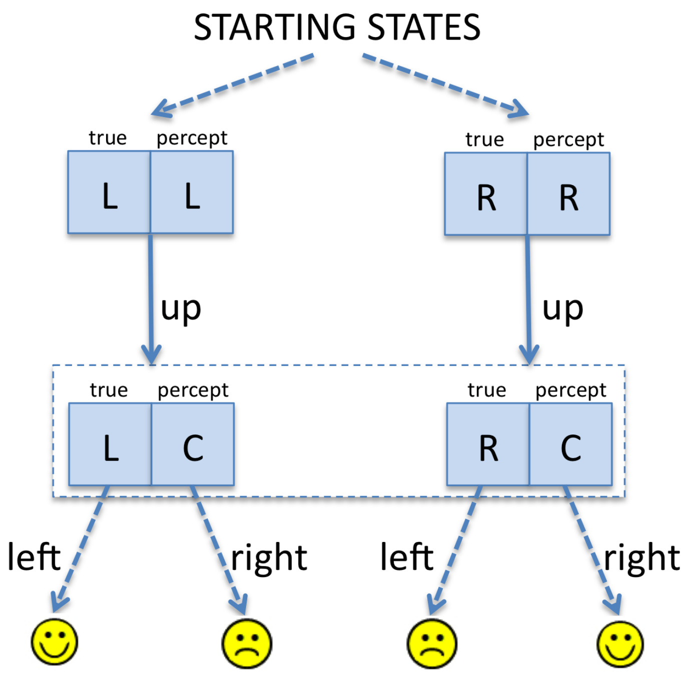

Figure 4.2: State diagram of a partially observable TMaze task.

- Exploration temperature
- Initial Q-values
- Eligibility trace decay rate

For the first three parameters, we conducted three-dimensional parameter sweeps across a discretized parameter space for various combinations of memory model and task. Agent performance for learning rate and exploration rate was relatively robust for a wide range of parameter settings, and by sampling a variety of tasks from our experimental corpus we set the learning rate to 0.2 and the temperature of our Boltzmann softmax action selection is 10^{-5}.

Observed performance, across all tasks and memory models, was very sensitive to initial Q-values. For the base TMaze and most other tasks, the best performing initial Q-values were 2.0; for tasks extended along the temporal delay and number of dependent action dimensions, the best performing initial Q-values were $2 + n$, where

n is the number of intermediate states or the number of dependent actions less one.

We made architectural design decisions in the same way, by exploring agent performance across a sample of memory model and task configurations and selecting those that resulted in the best performance. Pragmatically, this allowed for data to be collected more quickly. Performances of agents using Q-learning converged slightly more quickly than those using Sarsa, and thus we used Q-learning across all tasks. Similarly, moderating exploratory actions using a softmax procedure with a Boltzmann distribution converged more quickly than various epsilon-greedy procedures with decaying exploration rates.

Eligibility trace decay rates were selected with parameter sweeps over discrete values of decay rates; after a neighborhood of best rates was identified, we then ran a second sweep with a finer discretization in order to settle on the best value.

Unless otherwise noted, all experimental conditions were observed for one thousand agent trials, grouping performances into fifty bins, and then averaging across all agents to generate smoothed learning curves. For each plot that is displayed, trials were conducted for an order of magnitude more episodes than were plotted in order to ensure that agent performance did indeed converge to a stable value and didn't become unstable at a later point in time.

CHAPTER V

Base TMaze

In order to study the dynamics of memory and parameterized task dimensions, we must begin with a clear understanding of how an agent can learn to use internal memory in the base TMaze task. In this section, we describe our initial experiments with agents that learn to use bit and gated memory in the base TMaze task (Figure 3.5) and experiments with modifications to those memories to better understand the performances that we observed.

5.1 Bit and Gated Memory in the Base TMaze

We endowed agents with both bit memory and gated memory, and observed their performances in the base TMaze task. Figure 5.1 illustrates the mean reward per episode that the agents accumulated.

Surprisingly, the difference in observed performances between the two conditions is dramatic. An agent endowed with gated memory can learn to perform the task quickly (and thus learn to use memory), but an agent endowed with bit memory cannot learn to perform the task. In addition to the performances for agents using gated and bit memories, the performance of an agent with lesioned memory is presented. This agent has a single internal action available to it at all times, but the internal action does not change the state of the environment (or of lesioned memory). After orders

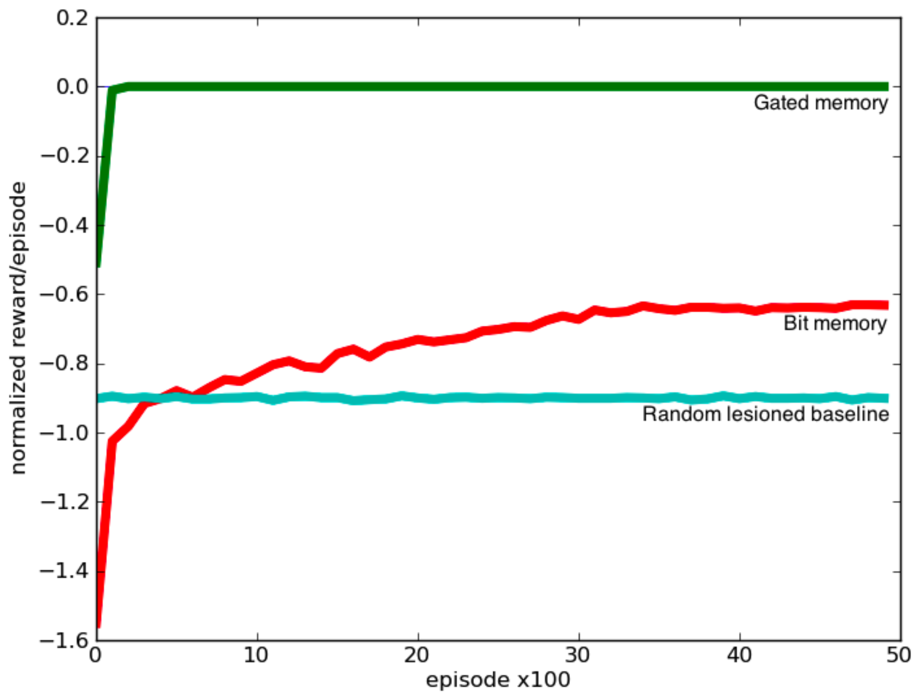

Figure 5.1: Performances of agents endowed with bit and gated memory in base TMaze task.

of magnitude more experience than an agent with gated memory requires, an agent with bit memory converges to an asymptotic performance that is only slightly better than that of the baseline condition of the lesioned agent.

5.1.1 Individual Behavior Analysis

The mean learning curves presented in Figure 5.1 paint a representative picture of the performances that agents using each memory exhibit, but averaging across performances sacrifices fidelity that could be helpful toward understanding why the differences between bit memory and gated memory are so stark. In order to understand the differences in performance, we examined the behaviors of twenty individual agents learning to use each memory model and qualitatively characterized the behaviors exhibited after four thousand episodes (sufficient to reach an asymptotic mean performance).

As expected from the plotted performances, agents using gated memory converged to the optimal behavior. In all twenty cases that we observed, agents learned to gate the salient symbol to memory, move forward in the task, and then select the correct terminal action.

In the twenty cases of agents learning to use bit memory, no agent converged to the optimal policy. In six cases, agents exhibited a stable behavior of toggling bit memory to a consistent value in the choice location and then selecting the same terminal action regardless of the salient observation; essentially, they learn to always set memory to the same value and then guess at a terminal action, but not to use memory to preserve history. In seven cases, a similar behavior was observed: the agents learned to set memory to a consistent value in the choice location, but alternated between which terminal action they select. For multiple episodes in a row, they would guess a single action, and then alternate and guess the other action for a number of episodes, and so on. Finally, seven agents were unstable and thrashed in the action space, guessing the correct terminal action half of the time and apparently having no stable behavior over internal memory.

Only in the seven thrashing, unstable agents did any agents select internal actions in the starting location. Agents that either converged to a stable behavior or to a mostly stable behavior only toggled the contents of bit memory in the choice location.

In prior work (Gorski & Laird, 2011), we identified a chicken and egg problem that can arise when agents learn to use memory: in order to converge to an optimal policy over how to control memory, an agent benefits from having found a stable behavior in the environment; simultaneously, in order to converge to an optimal behavior in the environment, the agent benefits from having found a stable policy for how to control its memory.

When an agent learns to use bit memory in the TMaze task, it encounters a similar problem: it must learn to select one of the external actions for one value of memory, and to select the other action for the other value of memory. What occurs instead is that an agent learns one of these associations first; for example, it learns to store 1 to memory when it perceives symbol A in the starting location, not to change memory when it is in the choice location and memory contents are 1, and also to select the correct terminal action in the choice location. After doing this a few consecutive times, the values for each of the corresponding state-actions begin to increase above the value of random actions, and the agent can begin to perform the optimal behavior when it observes A in the initial location.

Consider now what happens when an agent observes B in the initial location. An agent might set memory contents to 0 and then advance forward to the choice location. So far, this bodes well for the agent. However, when it is in the choice location it can take one of three actions: the correct terminal action, the wrong terminal action, and an internal action which will toggle memory contents to 1. Since the agent has not yet learned a stable behavior in this state, it will explore all of these options over time.

When an agent does try taking the internal memory action, it sets memory contents to 1. Now, the agent has learned a stable behavior for being in the choice location with 1 in memory: it should select the terminal action that corresponds to salient symbol A, because it has already learned that it is highly rewarding. But this is bad! In this case, the agent observed B, and so two things now happen.

First, the agent updates the value of being in the choice location with a 0 in memory and toggling memory contents with a good value, as the value of being in the choice location with memory contents of 1 and selecting the terminal action corresponding to A had high value. This is undesirable, because the optimal behavior is for the agent to never toggle memory in the choice location.

Second, the agent receives negative reward for selecting the wrong terminal action in the choice location. This means that what had been a good behavior is now negatively reinforced and the agent begins to unlearn the partially-optimal behavior that it had previously learned.

Through this example, it is clear why learning to use bit memory in TMaze is difficult, as the agent can very easily interfere with its own learning and essentially unlearn desirable behaviors.

5.1.2 State Diagram Analysis

While an agent using bit memory has considerable difficulty in the base TMaze case, an agent using gated memory is able to quickly learn both the task and how to use memory appropriately. A state diagram analysis, as introduced in Section 4.2.2, distinguishes the two conditions.

Figure 5.2 illustrates the state diagram for an agent using gated memory in the TMaze task. For this memory and task combination, a portion of the state space is completely observable and the agent can quickly learn an optimal behavior: gate memory in the initial location, move forward, and then select the appropriate terminal action. The trajectories through the state diagram that trace the optimal behaviors are shadowed in the Figure.

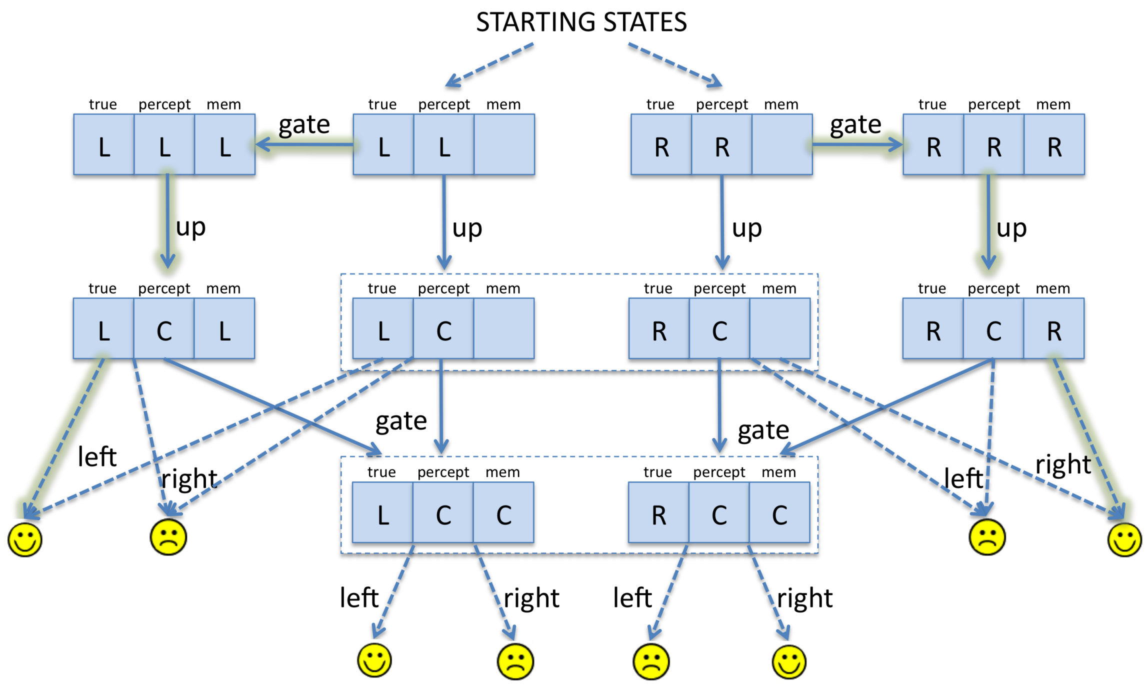

Figure 5.2: State diagram for an agent learning to use gated memory in the TMaze task.

For the states with terminal actions that are not aliased {LCL, RCR}, the agent learns that the value of the correct action is 1.0 and the value of the wrong action is -1.0. On the other hand, if the agent never gates memory, it will select a terminal action in one of two pairs of aliased states {LC_, RC_} or {LCC, RCC}, where the expected value for each action is 0.0. The agent finds itself in an equivalent situation if it gates memory in the choice location.

Figure 5.3 illustrates the state diagram for an agent using bit memory in the TMaze task. There are several important characteristics of the state-action space in the bit memory condition that distinguish it from the gated memory condition.

First, any trajectory to the optimal terminal reward traverses aliased states. States

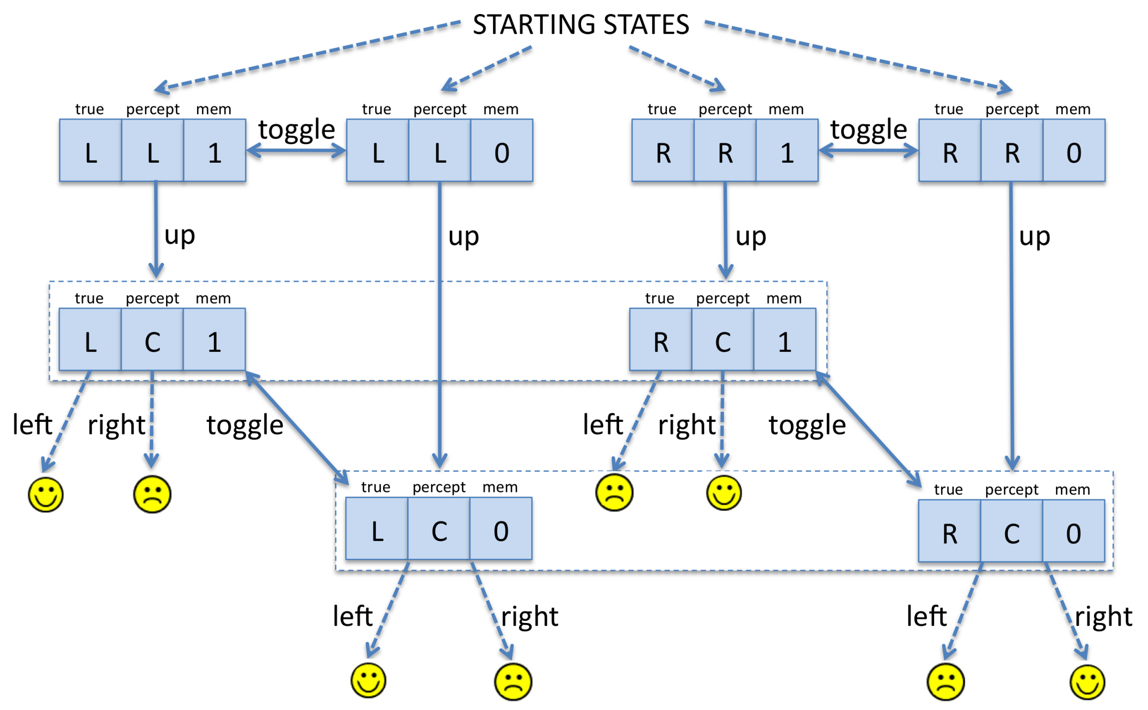

Figure 5.3: State diagram for an agent learning to use bit memory in the TMaze task.

that are aliased are those in which information in memory and the observation from the environment is not sufficient to disambiguate the true latent state of the system, and that there is uncertainty from the perspective of the agent as to which state it is actually in. The learning methods used by agents in our framework are guaranteed to converge in completely observable tasks (under circumstances related to how architectural parameters are controlled). However, in partially observable tasks with aliased states they lose these convergence properties, and agents are not guaranteed to be capable of learning to perform the task.

In contrast, optimal behaviors for agents using gated memory do not traverse aliased paths in the respective state diagram, and instead are completely disambiguated.

Second, the bit memory agent has an internal action available to it, toggle, that can transition between pairs of aliased states. It is this feature of the state diagram that leads to the self-interfering behavior described in Section 5.1.1. In this case, an agent learns an optimal behavior when observing one salient symbol; then while it is observing the other salient symbol an agent begins to interfere with its previously learned behavior by setting memory as if it had observed the other symbol instead.

The state diagram illustrates the behavior described in the individual analysis of the previous section. If the agent learns to only transition to states $\{LC1, RC1\}$ when it initially observes "L" and then selects the "left" terminal action, receiving positive reward, then when the agent finds itself in the states $\{LC0, RC0\}$ it will observe a positive reinforcement learning update for transitioning to those states, encouraging it to transition to those states and thus interfere with the correctly learned behavior.

The state diagrams for each condition make clear that there are significant differences between the effects of the two memory models. Gated memory, by storing observations directly, disambiguates the choice location because of what an agent could have possibly experienced – that is, an agent cannot store an observation that wasn't made in this task instance, and thus later states are disambiguated.

An agent with bit memory, on the other hand, must learn to associate a value of memory with an observation, and then not interfere with memory contents in successive states. Unfortunately for agents using bit memory, this creates a chicken and egg problem: the agent cannot settle on a stable memory control behavior without first having a stable environment behavior, and vice versa.

5.2 Architectural Modifications to Bit Memory

To better understand which aspects of bit memory were responsible for the differences in the observed learning performances, we modified the bit memory model (and the agent framework). In light of the state diagram analysis in the previous section, we hypothesized that if we prevented an agent using bit memory from selecting internal actions in the choice location that it might then settle on a correspondence between the value of internal memory and salient symbol.

The state diagram for this modified agent architecture is shown in Figure 5.4. While there is still no trajectory through the state space that is unaliased, an agent cannot transition between pairs of aliased states, which we hypothesized could ameliorate the self-interference behavior that it otherwise exhibits.

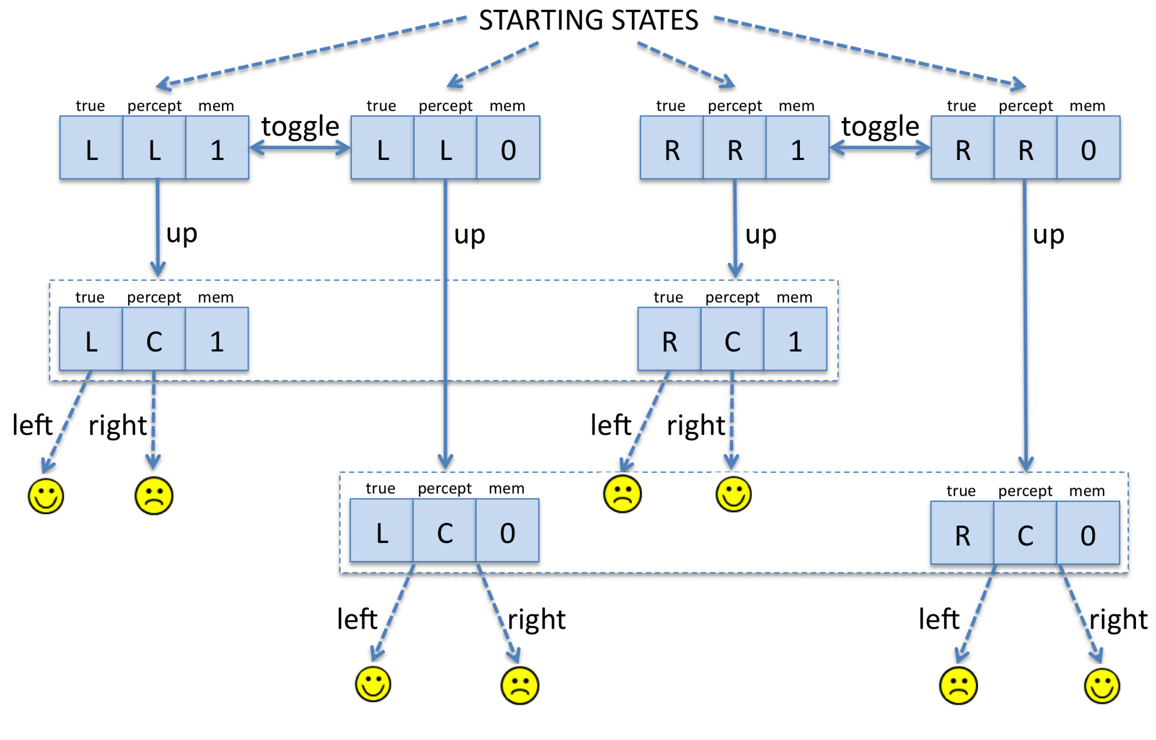

Figure 5.4: State diagram for an agent learning to use bit memory in the TMaze task, where the framework has been modified to prevent internal toggle actions in the choice location.

The second architectural modification that we explored was initializing bit memory to a special null value. We observed that gated memory has more representation power than bit memory: in the base TMaze task, it can store four possible values to memory (null, A, B, C); bit memory can only store two. We modified the bit memory model so that it was initialized to a special null value. We explored two different modifications; in one, the agent's toggle action transitioned from null to 0 and then behaved as normal; in the other, the agent had two *set* internal actions available rather than a single toggle action, each of which set internal memory to a specific value (for example, set_0 set the value of memory to 0).

Figure 5.5 illustrates the state diagram for null-initialized bit memory with set actions. Again, there is no trajectory through the state space that traverses only unaliased states, but in this state diagram an agent can still transition between pairs of aliased states. This contrasts with the previous state diagram (when toggle was prohibited in the choice location) and agents could not transition between aliased states.

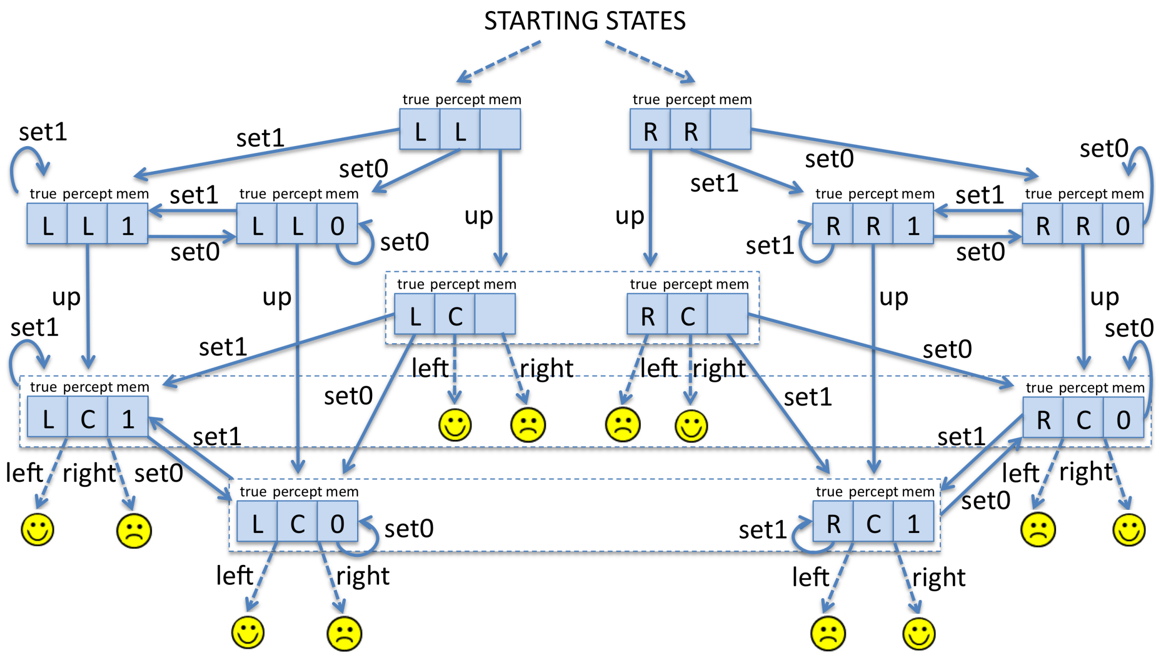

Figure 5.5: State diagram for an agent in the TMaze task learning to use null-initialized bit memory with set actions.

The results for these modified memory models (and agent framework) are shown in Figure 5.6. Agents that are restricted from toggling bit memory in the choice location converge to near-optimal behavior on average, albeit after orders of magnitude more experience than an agent using gated memory. Although the agent is prevented from interfering with partially learned behaviors in the choice location, it must still learn an association between salient symbols and values in memory; settling on a stable association requires significant experience, and is something that is provided architecturally to gated memory.

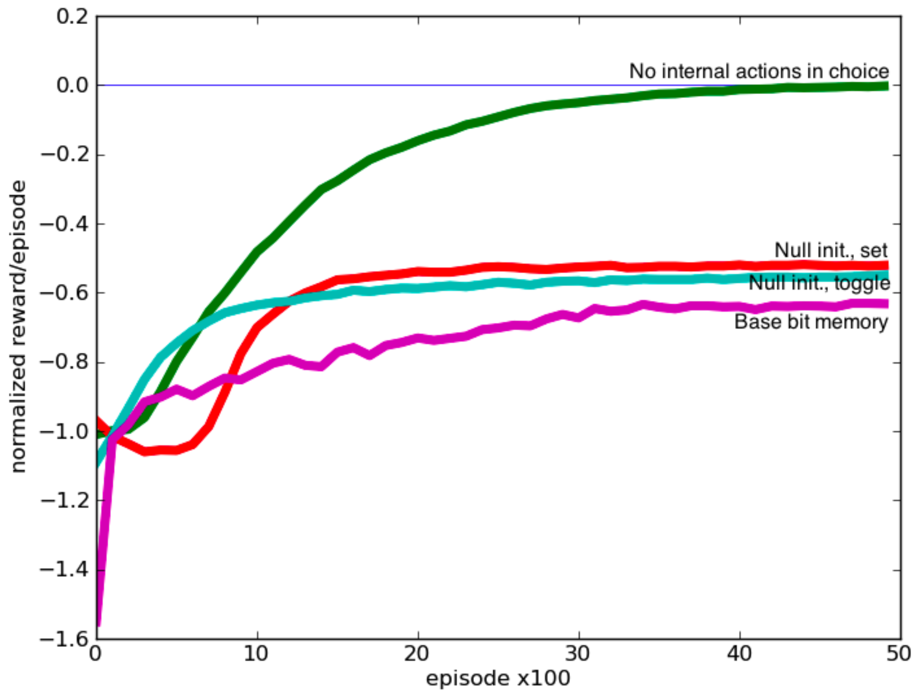

Figure 5.6: Restricting the bit memory toggle action and initializing bit memory to null.

Initializing bit memory to a special null value improves agent performance in the task over that of the unmodified bit memory condition. While it improves overall average agent performance, agents for both null-initialized memory conditions converge to behavior that is closer to that of the unmodified bit memory than optimal performance.

We conducted an individual behavior analysis for agents in each condition, identifying and categorizing the exhibited behaviors. We examined behaviors starting with the four thousand five hundredth episode, after average agent behavior was stable.

The condition in which agents were prevented from toggling the contents of bit memory exhibited the best performing agents, and converged to a near-optimal rate of observed reward (Figure 5.6). Our individual analysis found that, indeed, the agent converged to the optimal behavior in seventeen of twenty examined trials. In the other three trials, the agent exhibited a stable near-optimal behavior in which it learned the correct behavior for one of the salient observation symbols, but failed to act correctly for the other alternate salient observation. When observing the latter, an agent would immediately move up in the task regardless of the contents of memory; half of the time it will select the correct terminal action, and will select the wrong one half the time. The expected average reward for an agent exhibiting this behavior is 0.35 reward per episode.

In both of the conditions where agent memory was initialized to a special null value, the individual analysis showed that agents engaged in similar behavioral patterns. Table 5.1 summarizes the results of the individual analysis. There were three different categories of behaviors. In the "near optimal" case, agents exhibited stable behaviors that were nearly optimal except that for one of the two salient observations, the agent would select two internal actions instead of one. An agent exhibiting this behavior would expect to average 0.75 reward per episode.

Table 5.1: Summary of individual analysis of behaviors for agents endowed with null-initialized bit memory in the base TMaze task.

Condition	Near optimal	Near optimal but unstable	Stable, consistent, 50/50
Set	8 / 20	8 / 20	4 / 20
Toggle	7 / 20	9 / 20	4 / 20

In the "near optimal but unstable", the agent would exhibit the same near optimal behavior but every so often the agent's internal associations of terminal actions and memory contents would alternate. For example, the agent might act optimally for a dozen consecutive episodes, but then select the wrong terminal action in the choice location; on successive episodes, it would associate terminal actions with the opposite values as they had been associated with on the previous episodes, and continue to act optimally for a period of time before repeating the alternating behavior. An agent exhibiting this behavior would expect to average between 0.44 reward per episode if it alternated every half dozen episodes to 0.6 reward per episode if it alternated every dozen episodes.

In the “stable, consistent, 50/50” behavior, the agent would immediately move to the choice location, toggle memory in the choice location to a consistent value (either 0 or 1, irrespective of the salient observation that was made), and then select the same terminal action in each episode. On average, then, agents exhibiting this behavior would expect to receive $-.05$ reward per episode.

The individual behavior analysis for unmodified bit memory appears above, in Section 5.1.1.

5.2.1 Discussion

As the summarized results in Table 5.1 show, most agents that learned to use null-initialized bit memory exhibited near-optimal behaviors, while only twenty percent of the observed agents for each condition performed similarly to behaviors exhibited by agents in the base bit memory condition. Given the mean performances of the agents as shown in Figure 5.6, we were surprised that none of the agents managed to converge to the optimal behavior, but instead only learned near-optimal behaviors.

However, the improvement over the base bit memory condition is significant, and can be ascribed to the increased representational power of the modified bit memory mechanism. Behaving optimally, an agent uses internal actions to set memory to a certain value depending on the observation in the initial state. In all near-optimal behaviors that we observed, agents would set bit memory unnecessarily, and use the null value to its advantage. This improvement in performance under the null initialization conditions demonstrates that the additional representational capacity can be advantageous to learning, even when less representational capacity is sufficient to perform the task.

Mean agent performance very nearly converged to optimal when internal actions were inhibited in the choice location. By architecturally preventing exploratory behavior that can only be harmful or irrelevant (but never helpful), agents were able to learn to use bit memory to perform in the base TMaze task. This suggests that if agents could detect the presence of salient knowledge and when it must be used and prevent changing the contents of internal memory in the meantime, that they might better learn to use bit memory–and avoid the chicken and egg problem that can plague agents learning to use memory.

The modified bit memory models thus suggest two architectural solutions to surmounting the chicken and egg problem endemic to agents that learn to use memory. First, additional representational power can help agents learn by expanding the state-action space; this may help alleviate the problem, albeit not completely.

Second, preventing agents from interfering with knowledge in memory that is critical to performing in the task can help an agent to learn to perform the task optimally; preventing agents from interfering with task-critical knowledge in memory could take the form of expert knowledge or heuristics imparted to an agent in the form of architectural constraints.

An example of a potentially useful heuristic in TMaze tasks would be prohibiting internal actions when more than one external action is available. Much of the interference effects that we observe in TMaze tasks involve an agent interfering with the contents of internal memory in a choice location in order to "chase" a high Q-value, but in the process losing the knowledge that corresponded to the true underlying state of the task. Prohibiting actions in environment states that are choice locations would prevent the agent from engaging in these self-destructive interference behaviors and enable the agent to more easily learn to use memory.

5.2.2 Additional Architectural Modifications to Bit Memory

In addition to the modifications to bit memory described above, we also explored other modifications to bit memory. We summarize those experiments here but do not plot results or perform additional analyses.

We explored using set actions without null-initialized bit memory. In one case, we used two set actions; in the other, we used a 4-ary bit memory model with four corresponding actions. Both conditions underperformed base bit memory, with the 4-ary bit memory performing worse than the binary bit memory.

We also explored different memory initialization policies, thinking that it might affect initial behavior and thus learning performance. Instead of the default policy of initializing bit memory to each possible value with equal probability, we biased initialization so that one value would be selected with one hundred percent, seventy five percent, and sixty six percent probability. For each of those conditions, agent performance was the same as or worse than the condition when memory is initialized with equal probability.

5.2.3 Unmodified Memory Models with Eligibility Traces

In addition to architectural modifications to the memory models and restricting the space of available agent actions, we can also explore the effect of using eligibility traces in our reinforcement learning framework. Eligibility traces blend temporal difference and Monte Carlo updates, which also have the effect of updating more reward across more than one state-action-state transition. This can allow online agents to often learn effectively even in partially observable domains (Loch & Singh, 1998), as we discuss in Section 2.1.

As seen in state diagrams, agents learning to use gated memory can avoid aliased states and learn the optimal policy that traverses only un-aliased states. This lead us to hypothesize that eligibility traces should help agents learn to perform in the TMaze task when using bit memory, when all behavioral policies traverse aliased states. We thus explored learning to use memory with eligibility traces to determine how much it would improve learning to use bit memory.

Figure 5.7 shows the mean performances of agents that learn to use bit and gated memory with eligibility traces and an eligibility decay rate of 0.9, which was found to be optimal in a parameter sweep. Note that the scale of the horizontal axis differs in resolution from Figure 5.1 and illustrates learning performance across an order of magnitude fewer episodes.

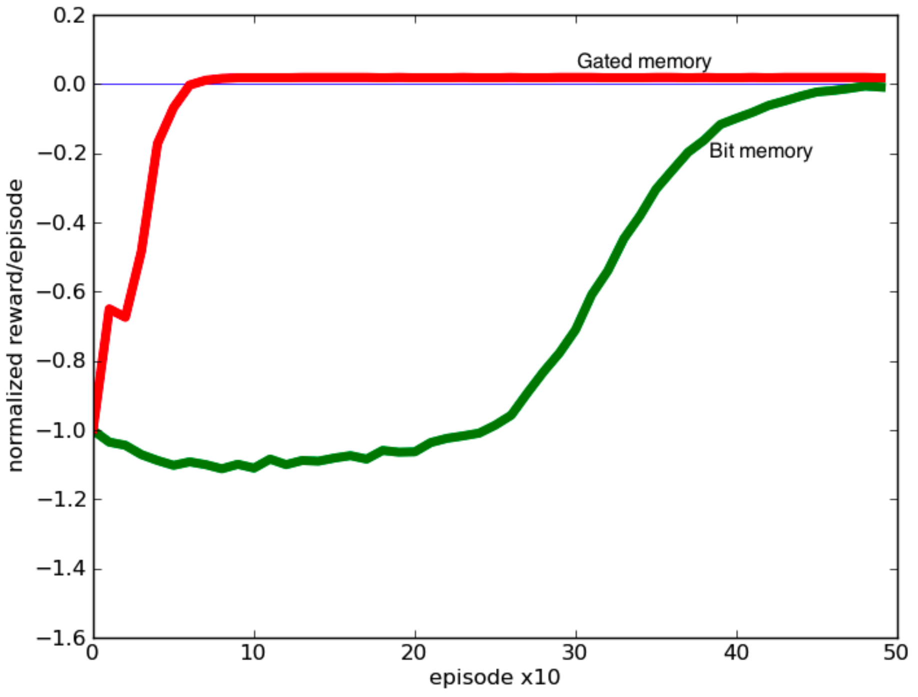

Figure 5.7: Agents learning to use bit memory and gated memory in the base TMaze task with eligibility traces.

It's clear that eligibility traces dramatically improve the performance of the bit memory condition. Bit memory converges to a near-optimal behavior by the five hundredth episode.

Agent performance in the gated memory condition also improves. While agents converge in roughly the same amount of experience, agents converge to behavior that, on average, is higher than optimal. An individual behavior analysis makes clear that gated memory agents learn to gate one salient observation to memory, but never gate memory when they observe the alternate salient symbol and instead condition behavior on the null-initialized contents of memory in that case. Such a behavior averages 0.85 reward per episode, and not all agents learn it.[1] This behavior was not observed in the individual analysis when eligibility traces were not used.

At this point, it behooves us to ask which results are more significant: those of agents using eligibility traces, or the earlier results of agents not using eligibility traces? This thesis aims to improve our understanding of the relationship of online reinforcement learning agents that learn to use memory to support performance in partially observable domains, and to that end we are agnostic to the choice of algorithm and instead are interested in more fundamental dynamics that arise. Therefore, *both* conditions are relevant insofar they give rise to qualitatively different performances, as they do in the base TMaze.

However, if we can evaluate learning to use memory for pure temporal difference agents without eligibility traces (as is the case with agents learning to use gated memory) then we will not investigate performance with eligibility traces. In these cases, agents learn sufficiently well so as to make trends readily apparent when learning without eligibility traces. It also assumes that using eligibility traces would not negatively impact learning performance. This assumption is valid for two reasons. First, these tasks are deterministic and the partial Monte Carlo backups that eligibility traces afford should not negatively impact time to convergence, they should only help for reasonable settings of the eligibility trace decay rate. Second, our methodology

[1]It is clear, then, that our usage of "optimal behavior" throughout this thesis is a misnomer. The alternative, however, is summarizing the same level of performance for near-optimal behavior in which an agent uses memory appropriately in both conditions but does not learn that the lack of knowledge of an observation in gated memory can be just as informative as that knowledge itself, which is clearly overly cumbersome. We kindly ask forgiveness from the reader for overloading the term optimal, and we will continue to use optimal in this way throughout the rest of the document.

specifies that we sweep the parameter space of the eligibility trace decay rate and select the best setting, which guarantees that we would never perform worse than pure temporal difference as we would select a decay rate of zero in the worst case.

We revisit the above question and answer it for other points of view in Section 11.5.

CHAPTER VI

Temporal Delay

6.1 Description of Temporal Delay

The TMaze task can be extended by introducing a delay between when the salient symbol is observed and when knowledge of that symbol must be used to make a decision. In the TMaze task, this delay takes the form of additional locations in which an agent must execute an action, but in which observations made and environmental actions taken are irrelevant to the terminal reward that the agent receives.

Figure 6.1 illustrates a TMaze extended along the temporal delay dimension. Locations marked as $X_1, ..., X_n$ are *intermediate* locations. When an agent is in an intermediate location there is only one environment action available; when selected in intermediate location X_i the action transitions the agent to location $X_i + 1$ if $i < n$ or to C if $i = n$.

When the TMaze task is extended along the temporal delay dimension, observations must be provided for the intermediate locations. There are two different configurations that can be used. In one, in each location the agent makes a unique observation, thus disambiguating each intermediate location. For example, in location X_i the agent would observe X_i. In the other configuration, all observations in intermediate locations are identical and thus each intermediate location is perceptually ambiguous. For example, in location X_i, the agent would simply observe X for

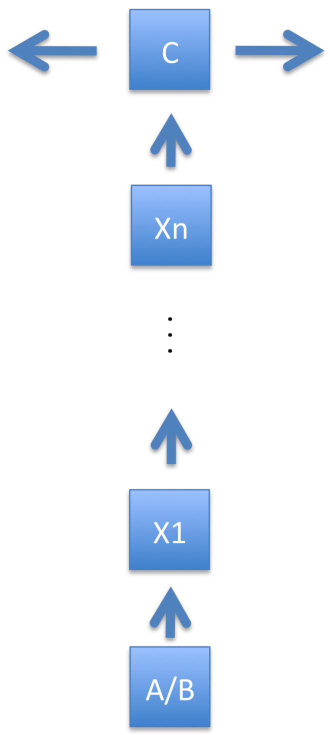

Figure 6.1: The TMaze task extended along the dimension of temporal delay.

all i.

In order to successfully perform the temporally extended TMaze, the agent must:

1. Encode and store knowledge of the symbol presented in the first location, either A or B.
2. For each intermediate location: move to the next location and not modify the contents of internal working memory (for example, not gate memory or toggle bit memory).
3. Move to the choice location.
4. Not modify the contents of internal memory before the next step (for example, not gate memory or toggle bit memory).
5. Associate knowledge of the salient observation with available actions and select

the correct action in the environment.

As the temporal delay increases, the task becomes more difficult for the agent. When extended with unique symbols, the agent must determine that all unique intermediate symbols are irrelevant to accomplishing the task, and maintain the true salient knowledge from when it is observed until selecting the correct terminal action. For identical symbols, the agent must traverse locations with repeating, identical observations while maintaining the salient knowledge in memory and then determining the correct correspondence between salient symbol and correct terminal action.

6.2 Temporal Delay Experiments

Our investigation into temporal delay seeks to answer the question of how an increasing delay between a salient observation and dependent action affects an agent's ability to learn to use memory. As discussed in Section 6.1, there are two types of TMazes extended along the dimension of temporal delay: one type where all observations made in intermediate locations are unique, and the other type where all observations made in intermediate locations are identical. We begin by presenting results for the unique observation case, and follow that with results for the identical case.

6.2.1 Unique Observations in TMaze Extended with Temporal Delay

As seen in the previous section, an agent endowed with gated memory can learn to perform in the base TMaze task very quickly. It is no surprise, then, that agents learning to use gated memory can also converge to optimal behaviors in temporally extended TMazes as well.

Figure 6.2 shows results for agents learning to use gated memory in TMazes extended along the dimension of temporal delay, where the number of intermediate locations is varied between zero and forty inclusive (individual learning curves are not labeled, but the curves converge in the order of the number of intermediate locations as would be expected).

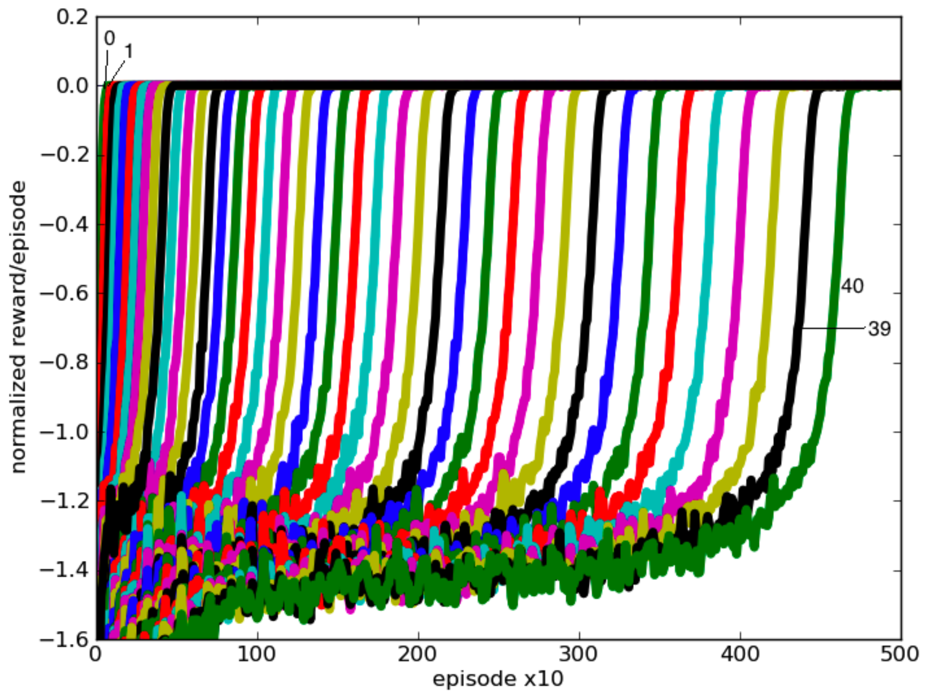

Figure 6.2: Agent performance when learning to use gated memory in the temporal delay TMaze task with unique observations, for zero to forty intermediate locations.

Note that although all learning curves are shown as converging to 0.8 reward per episode, this level of performance can only be achieved by an agent in the base TMaze task; every additional intermediate state results in an additional action that must be taken in the environment, and thus an additional -0.1 penalty that must be incurred. When displaying the results we thus add $0.1 * n$ to each point in a learning curve where n is the number of intermediate states in order to make it easier to visually compare performances across the dimension of task.

It is clear from the results that increasing the temporal delay between salient observation and dependent action does not negatively impact performance, and that the tasks remain tractable.

We plotted the time until convergence to optimal behavior[1] and found a quadratic fit to the time until convergence with an R^2 value of 0.99998, as seen in Figure 6.3. When agents learn to use gated memory, then, time until convergence scales quadratically with the number of locations between when salient knowledge is acquired and a dependent action that is conditioned upon it.

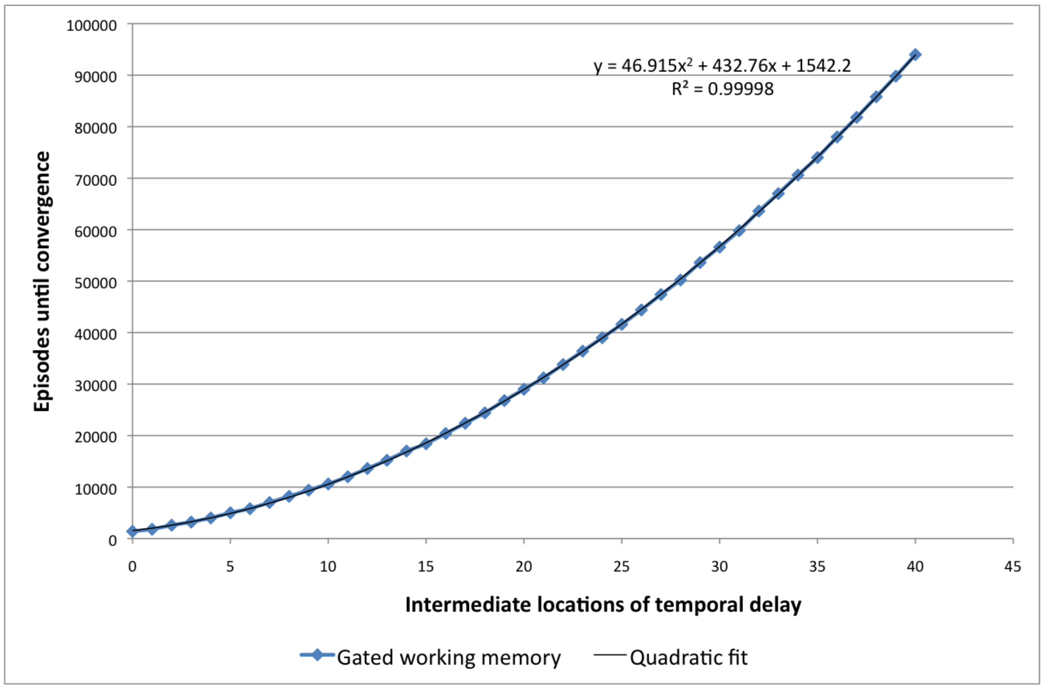

Figure 6.3: Number of episodes until convergence to optimal behavior for agents learning to use gated memory in the TMaze extended along the dimension of temporal delay.

As seen in the previous section, agents learning to use bit memory failed to learn near-optimal behaviors in the base TMaze; by extending the base TMaze along the dimension of temporal delay, and thus increasing the complexity of the task, there is

[1]Time until convergence to optimal behavior is defined as the first episode in which an agent is within 0.01 reward per episode of the optimal behavior and does not deviate from more than 0.01 reward per episode after that point. Rewards were averaged in buckets, effectively smoothing the learning curves across both agents and episodes; in this case, performances were grouped into five hundred buckets for analysis.

no reason to think that agent performance will improve.

Indeed, agents perform uniformly poorly as temporal delay increases. Although in the base TMaze agents learning to use bit memory average above 0.2 reward per episode, demonstrating that some learning is taking place, in tasks with any intermediate locations agents average -0.2 reward per episode (after normalizing for the number of intermediate locations) and demonstrate no improvement over time.

In the base TMaze, agents that learned to use bit memory in conjunction with eligibility traces quickly converged to the optimal behavior. The same is true as TMazes are extended along the dimension of temporal delay, as seen in Figure 6.4, although average asymptotic performance falls sharply as the degree of temporal delay increases. Agents exhibit near-optimal behavior for zero to one intermediate states, and as additional intermediate states are added to the task performance continues to degrade. These results have been optimized as were those presented in Figure 6.3.

We performed an individual behavior analysis on agents that were learned to use bit memory with eligibility traces, and a summary of the results are presented in Table 6.1. The trend in averaged learning performance is clear from Figure 6.4, and the individual analysis helps to explain why average performance suffers as the number of intermediate locations increases. In the base TMaze, all agents learn the optimal behavior. As the number of intermediate locations increase, more agents begin to converge to two suboptimal behaviors.

Table 6.1: Summary of individual analysis of behaviors for agents learning to use bit memory in the temporally extended TMaze task with unique observations.

No. intermediate locations	Optimal	3/4 correct	1/2 correct
0	100%	0%	0%
1	90%	0%	10%
2	50%	25%	25%
4	20%	15%	65%
8	0%	0%	100%

One of those suboptimal behaviors is labeled the “3/4 correct” behavior; in this

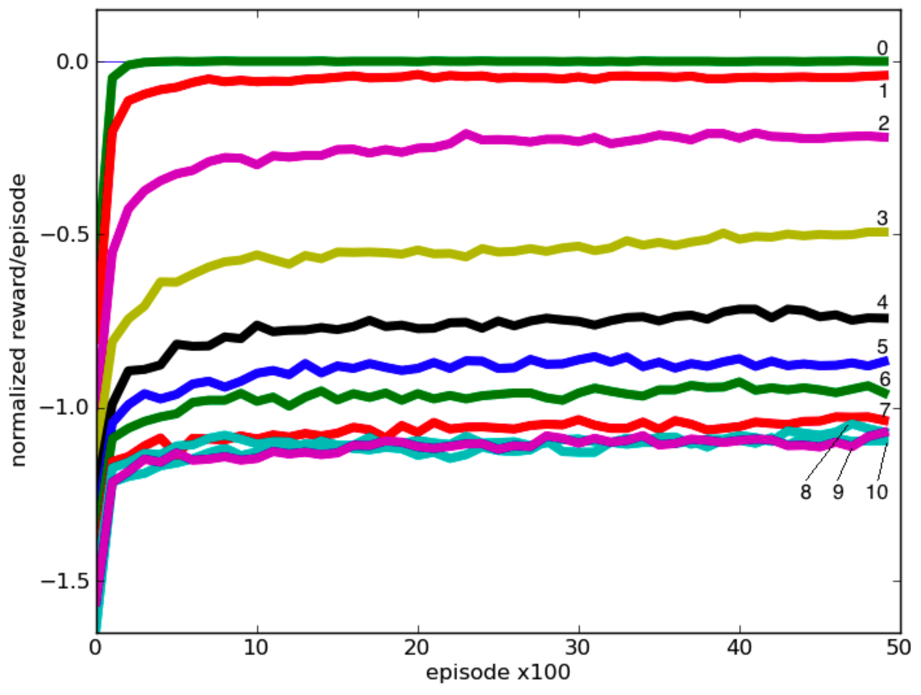

Figure 6.4: Agent performance when learning to use bit memory in the temporal delay TMaze task with unique observations, for zero to ten intermediate locations, and an eligibility trace decay rate of 0.9, normalized to present the difference from optimal behavior.

case, agents learn to correctly toggle bit memory for one value of salient observation, but for the other salient observation never select internal actions. For example, an agent might learn to set internal memory to 1 when it observes A, but when it observes B it does not toggle memory regardless of memory contents. In the choice location, it then selects the correct terminal action for the case in which it observed A, but when it had previously observed B it can only correctly select the terminal action 50% of the time.

The other observed suboptimal behavior, labeled "1/2 correct", is when the agent learns to select one terminal action always and never selects the alternative. Whereas in previous individual analyses we observed unstable and changing behaviors, in this analysis all behaviors were stable and unchanging.

Thus, even with eligibility traces agents that learn to use bit memory become increasingly unlikely to learn optimal behaviors as the number of intermediate states increase. There are several reasons why agents become less likely to converge to optimal behaviors as the number of intermediate states increases:

- The probability of an agent selecting an internal action increases with the number of intermediate states. The exact probability depends on the estimated Q-values at a particular point in time, but if all values are identical and actions are selected with equal probability then the probability of not selecting any internal actions in intermediate locations is 0.5^n.

- Even with a high eligibility trace decay rate of 0.9, as the number of intermediate locations increases, the update that the earlier Q-values receive becomes smaller: the backup reaching an internal action in the first location is proportional to $\alpha * \lambda^{n+1}$, where α is the learning rate (0.2), λ is the eligibility trace decay rate, and n is the number of intermediate locations.

- Irrespective of the probabilities of the two above items, agents will still explore the entire state-action space in the task; only after repeatedly updating the wrong state-action values would it be useful to see traces of the optimal behavior. However, at that point they are unlikely to occur and increasingly less likely as the number of intermediate states increases.

In summary, when the TMaze task is extended along the dimension of temporal delay and intermediate states result in unique symbols as observations, agents learn to use gated memory and the task is tractable; agents learn to use bit memory in conjunction with eligibility traces for very simple tasks and then performance degrades quickly as the number of intermediate states grow.

6.2.2 Identical Observations in TMaze Extended with Temporal Delay

When observations in the intermediate locations of a temporally extended TMaze are identical, all of the intermediate states are perceptually aliased. Before running experiments, we were unsure how agent performance would change from the unique observation setting in the previous section to the identical observation setting. On one hand, all of the intermediate locations are aliased, and state aliasing does increase the difficulty of the learning problem. On the other hand, the optimal behavior in each location is the same: move up in the environment and do not select any internal actions; furthermore, the state aliasing allows the agent to generalize across the intermediate locations and thus could potentially improve learning performance, too.

The results of agents learning to use gated memory in TMazes extended along the temporal delay dimension with zero to ten intermediate states and identical observations are shown in Figure 6.5 (again, results are normalized such that performances are difference from optimal). The results look very similar to those of bit memory with eligibility traces in the unique observation case, above (Figure 6.4) – very quick convergence for all amounts of temporal delay, but agents learning to perform in TMazes with temporal delays of two and larger exhibit increasingly sub-optimal behaviors.

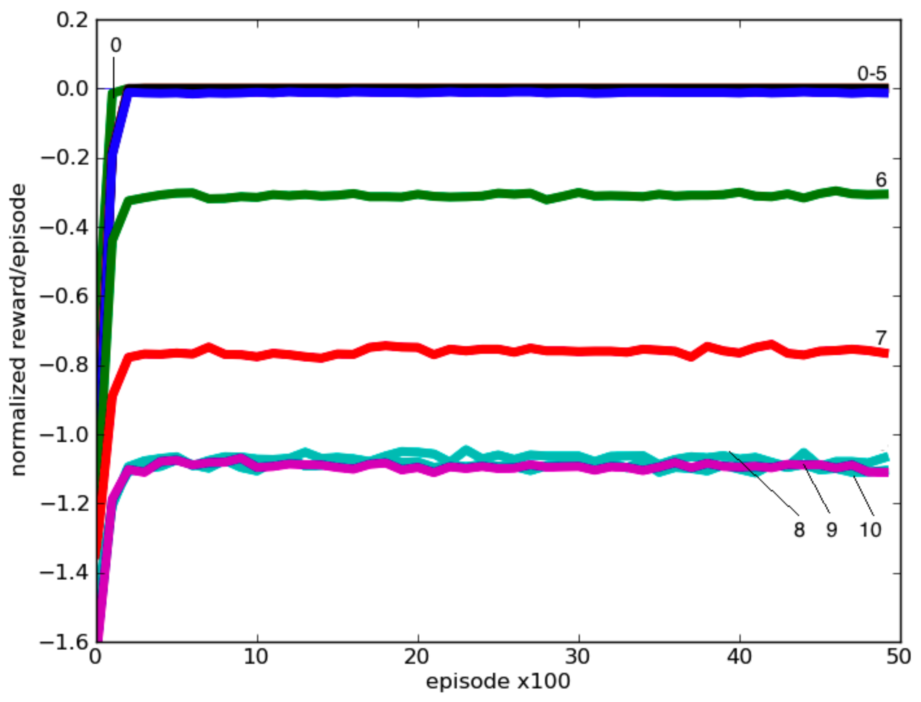

Figure 6.5: Agent performance when learning to use gated memory in the temporal delay TMaze task with identical observations, for zero to ten intermediate locations.

This is the first set of gated memory results in which mean agent performance did not converge to optimal, and as such, it is also our first opportunity to perform an individual analysis to examine the distribution of individual behaviors that agents exhibit. Table 6.2 summarizes the individual behaviors that agents exhibit across various numbers of intermediate states.

When we performed individual analyses of agents learning to use bit memory in the base TMaze task, agents were self-interfering with behaviors that were optimal for part of the task. For example, an agent would learn to set bit memory to 1 when observing A, and then take the corresponding correct terminal action. But the

Table 6.2: Summary of individual analysis of behaviors for agents learning to use gated memory in the temporally extended TMaze task with identical observations.

No. intermediate locations	Optimal	Opt., extra internals	1/2 correct
0-5	100%	0%	0%
6	40%	40%	20%
7	0%	20%	80%
8	0%	0%	100%

agent, after observing B, would toggle memory to 1 in the choice location because that Q-value would be relatively high, and thus interfere with its previously learned behavior.

When agents learn to use gated memory in the temporally extended TMaze tasks, the issue simply seems to be that as the delay grows beyond five intermediate states agent behavior stabilizes while still remaining sub-optimal. However, there is a significant difference between agents learning to use memory in this condition and the performances of agents that we observed learning to use memory in the unique observation condition. In the unique observation condition, time to convergence varied with the number of intermediate states. Here, in the identical observation condition, convergence happens at the same time regardless of the number of intermediate locations.

Although the number of latent environmental states varies, the number of unique observations that an agent experiences in the course of an episode is constant across conditions. The time to convergence, then, likely can be ascribed to the Q-value updates being performed by the agent's reinforcement learning mechanism. We will revisit this discussion below, when discussing results of agents that learn to use bit memory in the same tasks.

Having found a task in which the performance of agents learning to use gated memory is similar to performances that we have observed for bit memory, we explored two additional conditions: gated memory with eligibility traces, and gated memory when the gate action is inhibited in the choice location. We predicted that both conditions would result in convergence to optimal behavior.

Indeed, the results for agents learning to use gated memory with eligibility traces, shown in Figure 6.6 show that agents converge to optimal behavior relatively quickly, across zero to ten intermediate states. Note the large gap in time to convergence between the learning curves for one and two intermediate states. When there is one intermediate state, there is no aliasing (the *identical* observation is not truly identical as there is only a single intermediate state, and it isn't aliased with either the salient observation or the choice location). Thus, the aliasing that is introduced with two intermediate states makes the task significantly more challenging; additional state aliasing only slightly more challenging. The differences in performances between one and two intermediate state tasks can be interpreted as the cost of aliased states, which in terms of time to convergence, is at least an order of magnitude more experience.

As was the case with agents learning to use bit memory in the base TMaze, inhibiting internal actions in the choice location does result in convergence to optimal behavior, as seen in Figure 6.7. Although agents converge to optimal behaviors, they do not do so as quickly as agents that use eligibility traces. Furthermore, there is no gap between tasks of one and two intermediate locations.

As agents that learned to use bit memory struggled in the base TMaze task and the temporally extended TMaze task with unique observations, one would not expect them to perform well in a temporally extended TMaze with identical observations. Indeed, mean agent performances are poor, as seen in Figure 6.8.

It is intriguing that the performance of agents in tasks with one and two intermediate locations are nearly identical, and qualitatively different from the performances of agents for three and more intermediate locations. In the results for gated memory (Figure 6.6) we observed that there was a significant difference between tasks with one and two intermediate locations as the task with two intermediate locations is the first

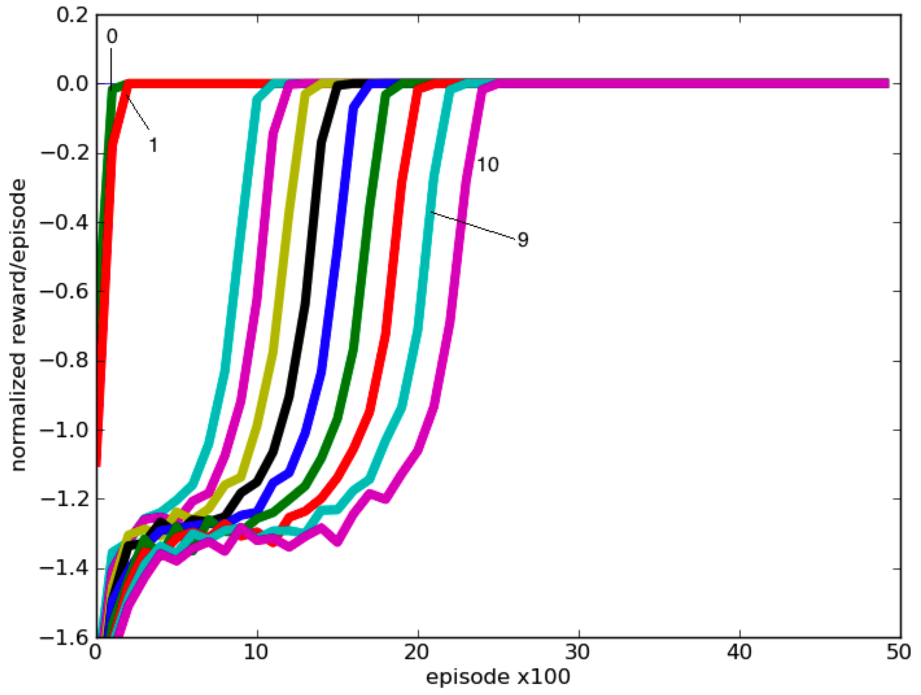

Figure 6.6: Agent performance when learning to use gated memory in the temporal delay TMaze task with identical observations, for zero to ten intermediate locations and an eligibility trace decay rate of 0.75.

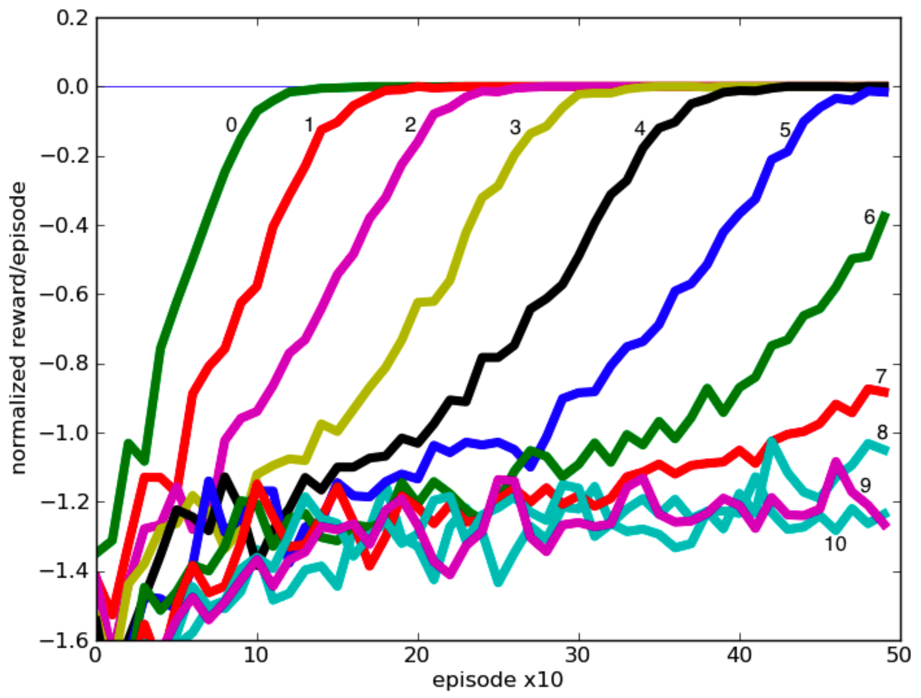

Figure 6.7: Agent performance when learning to use gated memory in the temporal delay TMaze task with identical observations, for zero to ten intermediate locations, when internal actions in the choice location are inhibited.

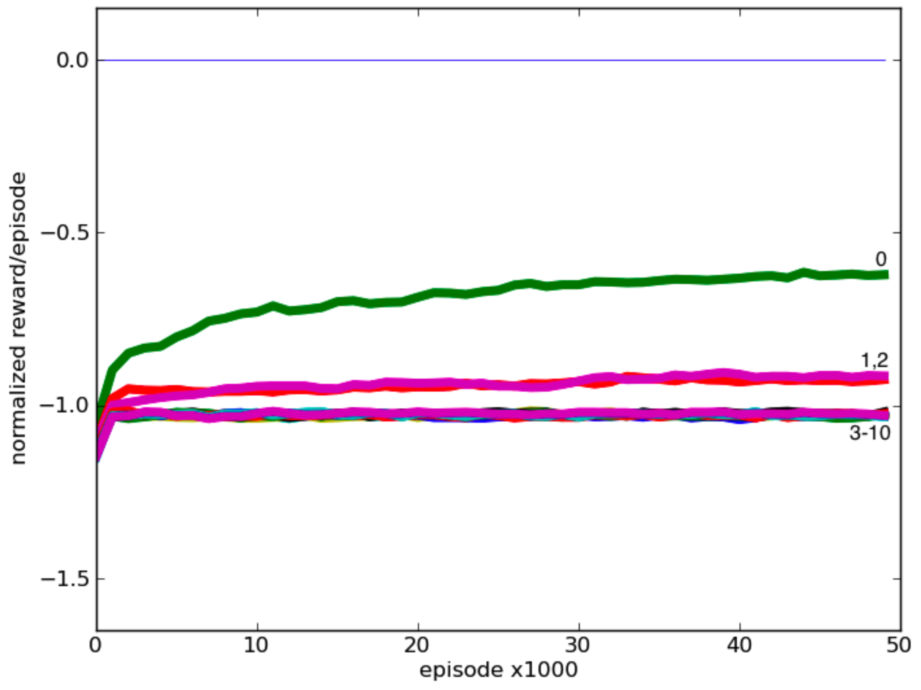

Figure 6.8: Agent performance when learning to use bit memory in the temporal delay TMaze task with identical observations, for zero to ten intermediate locations.

in which the intermediate locations introduce additional state aliasing. Why would the difference between one and two locations be indistinguishable for bit memory, but yet differ from results in tasks with three and more intermediate locations?

The individual analysis of behaviors exhibited by agents in these tasks, summarized in Table 6.3, confirms that there is indeed a difference in the behaviors that they are exhibiting. For one and two intermediate locations, agents occasionally (20% and 30% of the time respectively) converge to behaviors in which they select the correct terminal action three quarters of the time. This means that they correctly set internal memory for one salient observation, and for the other they do not change memory at all. Furthermore, they do not change internal memory in the intermediate or choice locations, thus averaging optimal behavior 75% of the time. On the other hand, when there are three intermediate locations (and more), agents never converge to this behavior and instead only select the correct terminal action 50% of the time.

Table 6.3: Summary of individual analysis of behaviors for agents learning to use bit memory in the temporally extended TMaze task with identical observations.

No. intermediate locations	3/4 correct	1/2 correct
1	20%	80%
2	30%	70%
3	0%	100%

Figure 6.9 shows results for agents learning to use bit memory in the same set of tasks but when using eligibility traces with a decay rate of 0.9. Again, agents exhibit similar performances in tasks with one and two intermediate locations; for three and more locations, agents continue to perform worse as the number of intermediate locations grows. This phenomena of qualitatively similar behaviors for one and two intermediate locations persists both when eligibility traces are and are not used, and is confirmed with an individual behavior analysis. This phenomena was not present when TMaze tasks were extended along the dimension of temporal delay with unique

observations; rather it is unique to the identical observation setting.

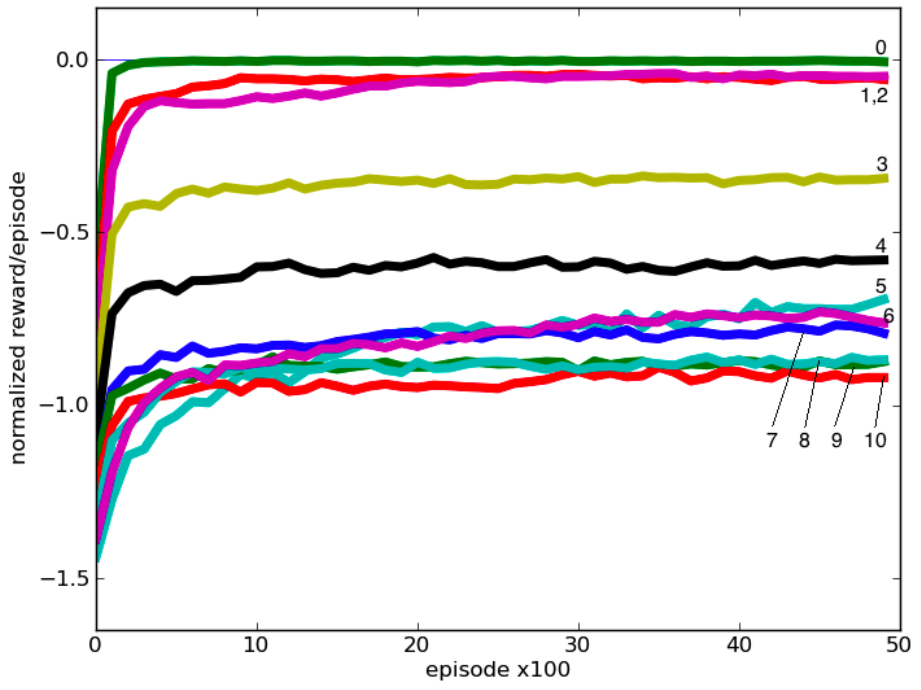

Figure 6.9: Agent performance when learning to use bit memory in the temporal delay TMaze task with identical observations, for zero to ten intermediate locations, in conjunction with eligibility traces and an eligibility trace decay rate of 0.9.

This phenomena is similar to performances that we observed when agents learned to use gated memory in the temporally extended TMaze tasks with identical observations. In performances illustrated in Figure 6.5, agents performed nearly identically across tasks with zero to five intermediate locations, even converging in almost identical amounts of time. This contrasts with agents that learned to use gated memory in tasks with unique observations in the intermediate locations, or agents that learned in conjunction with eligibility traces.

When additional aliased locations are added between when a salient observation is made and dependent actions are taken, it appears that there is a cliff in learning behaviors. For a certain number of locations or fewer, learning is tractable and agents can learn to perform the task. Beyond that cliff, agent performance suffers and does not converge to optimal, although it *does* converge in nearly the same amount of time. When agents do converge to a stable behavior, however, the ratio of optimal to sub-optimal behaviors decreases as additional intermediate locations are added until the ratio is zero.

Why might this cliff exist? While these agents use eligibility traces, they are still online agents and must balance exploration of their action space with greedy actions. As temporal delay increases, the probability of agents selecting the correct sequence of actions decreases; it's possible that agents simply do not observe sufficient traces of the optimal behavior in order to avoid converging to stable, suboptimal behaviors.

6.3 Summary of Temporal Delay Experiments

There were several trends that are apparent when looking at all of the results that we presented for agents learning to use memory in TMazes extended along the dimension of temporal delay. We can qualitatively group the behaviors that are exhibited across tasks.

In one group, all agents converge to optimal behaviors and the time to convergence scales quadratically with the number of intermediate locations. For agents to exhibit this behavior, they must converge to the optimal behavior in the base case; for example, gated memory converges to optimal in the base case while bit memory does not. We call this the *time-to-converge* group, as additional intermediate locations effects the time it takes an agent to converge to optimal.

In another, the base case converges to optimal behavior and adding intermediate locations degrades the asymptotic mean performance. As additional intermediate locations are added to the task, agents converge to more suboptimal behaviors and fewer agent traces (as found in individual behavior analyses) converge to the optimal, or near-optimal, behaviors. These stable, sup-optimal behaviors are easier to find via exploration and as additional locations are added to the task, the agent is less likely to find them before converging to a stable behavior. We call this the *ratio* group, as adding intermediate locations changes the ratio of optimal, near-optimal, and sub-optimal behaviors that agents converge to.

Finally, in the third case, the base case is sub-optimal and adding intermediate locations quickly causes agent behavior to degrade to the same uniform suboptimal behavior, in which the agent sets memory contents to one consistent value and select the same terminal action regardless of the salient observation made in the starting location. This case is similar to the second, but the fact that behavior in the base case is so poor makes it impossible to determine the effect of additional intermediate locations. We call this the *indistinguishable* group, as the performances of most agents after the base case are indistinguishable from each other (and no better than guessing).

Why do agents converge to three different types of stable behaviors, given that they initially start with identical states? There are several sources of nondeterminism (or noise) in our experiments:

1. Random exploration actions: while weighted towards greedy actions according to a softmax distribution, action selection is stochastic.

2. Distribution of task type: the salient observation in a task episode, and thus the correct terminal action, is randomly selected at the beginning of an episode with uniform probability.

3. Initialized Q-values: a small amount of noise is injected into agents' initial Q-values.

These sources of stochasticity are sufficient to lead to qualitatively different stable, convergent behaviors. Given that there agents converge to qualitatively different stable behaviors suggests that if an agent experienced particular sequences of task episodes (or actions within an episode) it might transition from one stable behavior to another, although we did not pursue this direction.

We summarize the results that were presented in this Section in Table 6.4, and categorize them according to the qualitative descriptions above. For each entry in the table, we list whether observations were unique or identical, which memory model was used, and any experimental conditions that were explored (where *base* indicates only the unmodified base agent framework). Agent behavior is then categorized according to the descriptions above, along with any notes.

Table 6.4: Summary of results in TMazes extended along the temporal delay dimension.

Observations	Memory	Conditions	Behavior group and comments
Unique	Gated	Base	Time-to-converge. Scales quadratically with number of intermediate locations.
Unique	Bit	Base	Indistinguishable.
Unique	Bit	Eligibility traces	Ratio.
Identical	Gated	Base	Ratio. Although 0-5 intermediate locations show identical performance.
Identical	Gated	Eligibility traces	Time-to-converge. Significant difference between 1 and 2+ intermediate locations.
Identical	Gated	Inhibit in choice	Time-to-converge. Slower convergence than eligibility traces condition.
Identical	Bit	Base	Indistinguishable. Although 1-2 intermediate locations are slightly better than 3+.
Identical	Bit	Eligibility traces	Ratio. 0-2 intermediate locations perform near-optimally.

CHAPTER VII

Number of Dependent Decisions

7.1 Description of Number of Dependent Decisions

The TMaze task can be extended by introducing additional decisions (and hence actions) that depend on knowledge of the salient observation. This involves increasing the number of locations after the (base) choice location, where each additional location also has two possible actions whose effects depend on the salient observation.

Figure 7.1 illustrates adding an additional dependent decision, for a total of two dependent decisions and one more than in the base TMaze. In this case, the location marked as C_1 has two actions; like the base TMaze, one action is correct when A is the salient observation in the first location and the other is correct when B is the salient observation. In either case, both actions lead to a location marked as C_2, where again there are correct and wrong actions in the same manner as in the C_1 location. After the terminal action is selected, an agent receives positive reward iff the agent has selected only correct actions; if the agent selected any action in a choice location that was not correct, it receives negative reward.

As additional dependent actions are introduced, the same characteristic of task holds in that an agent must select only correct dependent actions in order to receive positive reward. At any point, selecting an incorrect action will result in negative terminal reward.

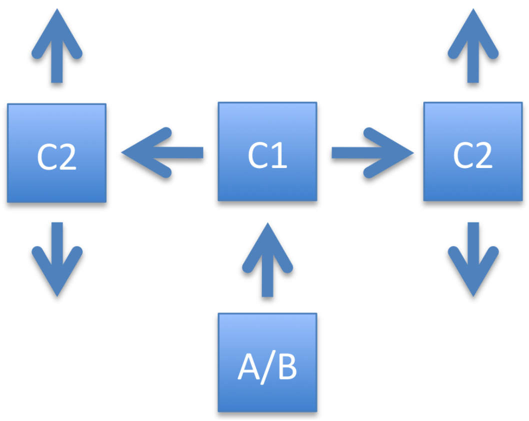

Figure 7.1: The TMaze task extended along the dimension of number of dependent decisions, for a total of two dependent decisions.

As was the case for temporal delay in Section 6.1, there are two types of TMazes with additional dependent actions: unique and identical. In the unique case, an agent observes a unique symbol in each location, such that the observation made in each location captures the history of environmental actions that the agent has selected so far in the current task episode. Continuing the example of a TMaze task with two dependent decisions, in the first choice location the agent would observe C_1. If the agent selected action A, it would then observe C_A; if it selected action B, then it would observe C_B.

In the second type of task with identical observations, the agent observes symbols that indicate how many dependent decisions have been made thus far in the episode but cannot recover action history from them. At the first choice location, then, the agent observes C_1; after selecting an action, it then observes C_2 regardless of which action it selected in the first choice location.

In order to successfully perform the TMaze with additional dependent decision, an agent must:

1. Encode and store knowledge of the symbol presented in the first location, either A or B.

2. Move to the first choice location with dependent actions.

3. While the agent is in a choice location with dependent actions: do not modify the contents of internal working memory, associate the salient knowledge with the correct action, then select that action in the environment.

As the number of dependent actions in a task increases, the task becomes more difficult. This is true for both types of tasks. In the unique observation case, the agent must learn to use memory correctly and to correctly act in each location with a dependent decision, conditioning the selected actions on the contents of memory. In the identical observation case, the agent must also overcome the disadvantage of not being able to recover the history of past actions while learning from environmental observations, and must learn sequences of correct decisions.

7.2 Number of Dependent Decisions Experiments

The purpose of our experiments in the dimension of the number of dependent decisions is to determine how adding additional decision points to a task affects an agent's ability to learn to use memory, where the correct action at a decision point is determined by a previously observed salient symbol. As discussed in Section 7.1, there are two configurations of TMaze tasks extended with additional dependent decisions: the unique and identical configurations.

We begin by presenting experiments and results in the unique observation case, and then proceed to present results from the identical case.

7.2.1 Number of Dependent Decisions with Unique Observations

Extending a TMaze with additional decisions that depend on a salient observation shares some similarities with the previous section, where we extended a TMaze with temporal delay. When the base TMaze is extended with a second dependent decision, the second dependent decision has a longer temporal delay between its location and the location in which the salient symbol is observed than the first dependent decision does; furthermore, the agent must learn the correct action in that second dependent decision. Therefore, we expect agent learning performance in these tasks to scale less favorably than it did along the dimension of temporal delay.

Figure 7.2 shows the results for agents that learn to use gated memory in TMaze tasks that are extended with additional dependent decisions and unique observations, for one to ten dependent decisions. By the five thousandth episode, mean agent performance has converged to optimal in tasks with one to five dependent decisions; the remaining conditions require more experience to converge. These results are not plotted, but all ten learning curves do converge to optimal.

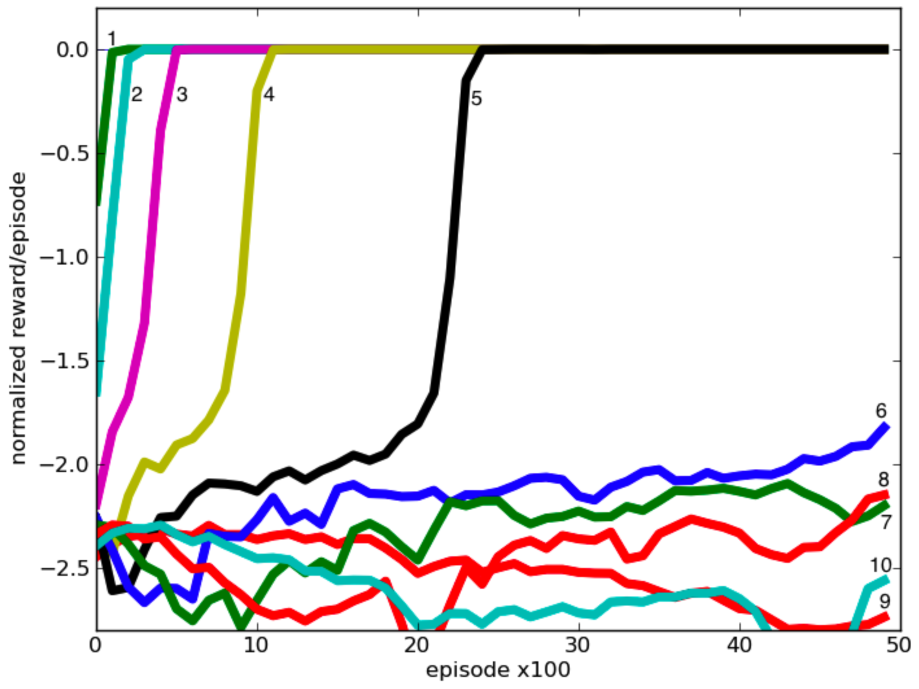

Figure 7.2: Agent performance when learning to use gated memory in the TMaze task extended with additional dependent actions and unique observations, for one to ten dependent actions.

Agent learning performance, then, does scale worse as TMazes are extended with additional dependent decisions than it did when temporal delay was added, as we expected. As a point of comparison, agents that used gated memory to perform in the TMaze task with ten additional locations of delay and unique observations had converged to optimal by the five hundredth episode (as seen in Figure 6.2).

We plotted the number of episodes required for convergence to optimal in Figure 7.3. The vertical axis is on a logarithmic scale, and an exponential regression was fit to the data with an R^2 value of 0.9973. Whereas the amount of experience required to converge to optimal scaled quadratically as TMaze tasks were extended along the temporal delay dimension, as tasks are extended with additional dependent decisions the amount of experience required to converge to optimal when learning to use memory scales exponentially.

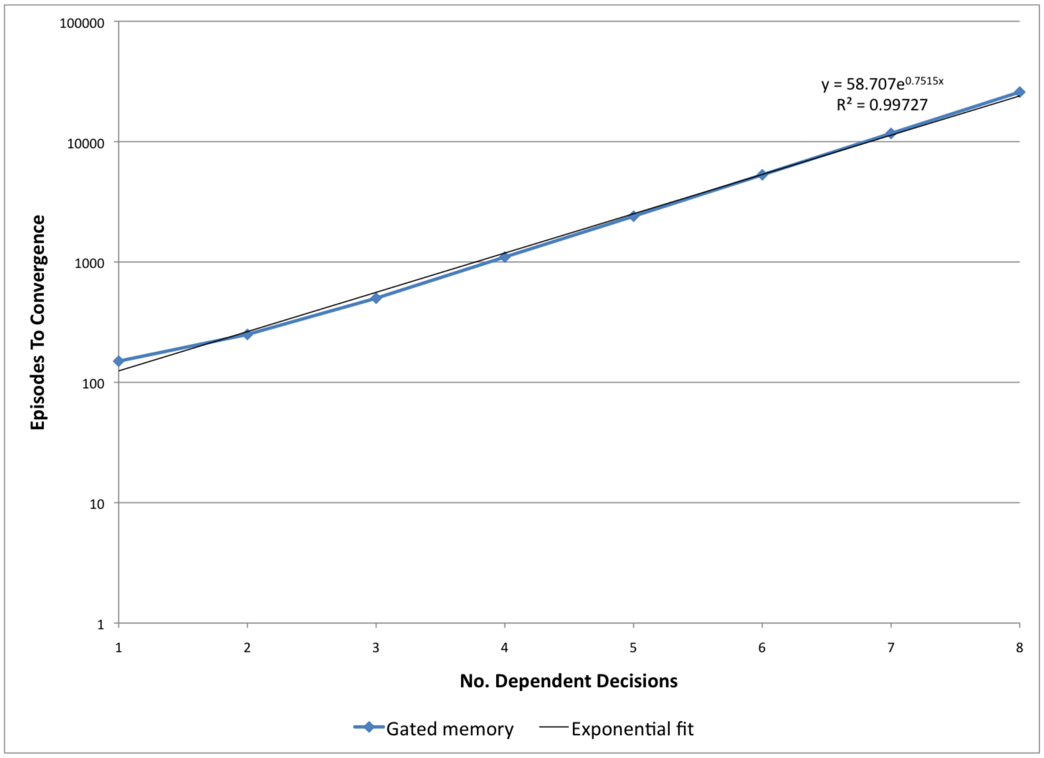

Figure 7.3: Number of episodes until convergence to optimal behavior for agents learning to use gated memory in TMaze tasks extended with additional dependent decisions.

A state diagram analysis is illustrated in Figure 7.4 for an agent using gated memory in a TMaze task with two dependent actions and unique observations.

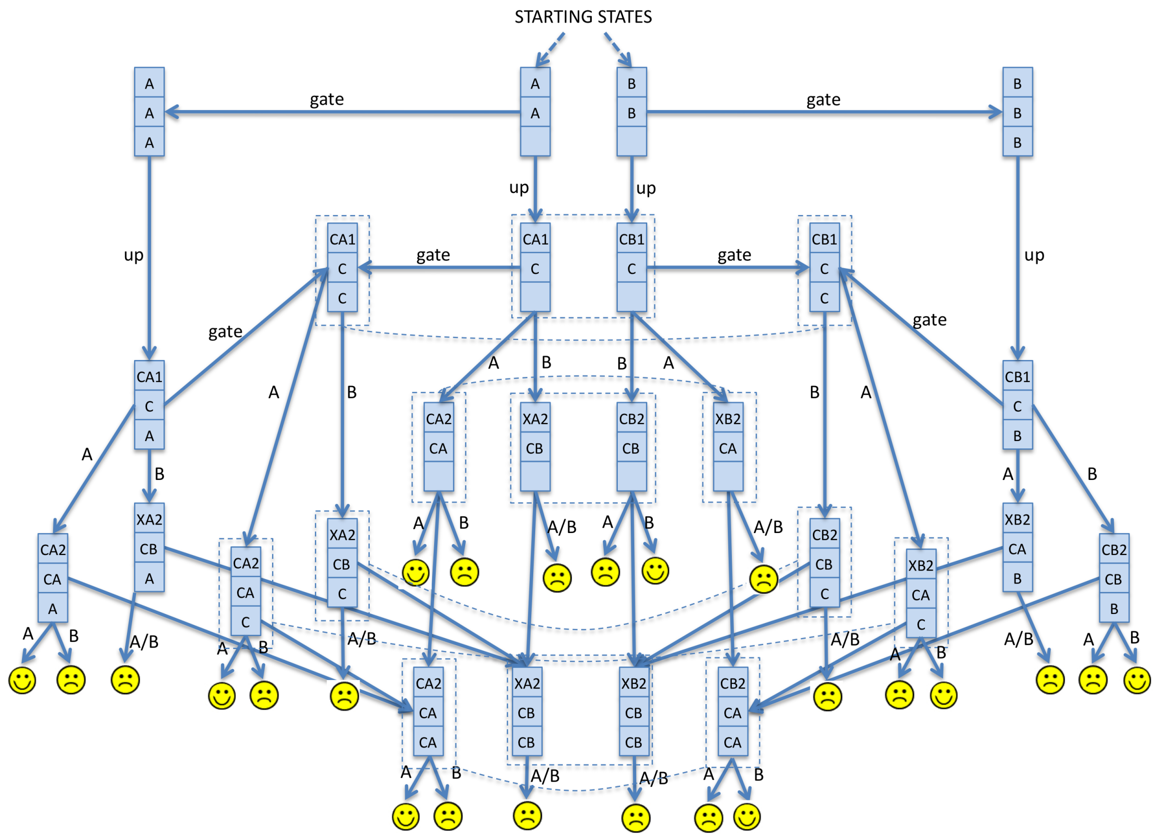

Figure 7.4: State diagram for an agent endowed with gated memory performing the TMaze task with two dependent choices and unique observations.

In this state diagram, there are trajectories through the state space that cover the optimal behavior and do not traverse any aliased states. Thus an agent learning to use gated memory in the TMaze task extended with additional dependent decisions would be expected to converge to an optimal behavior. We continue our discussion of features of the state diagram in Section 7.2.2, where we contrast it with the state diagram for the identical observation case.

Given that agents learning to use gated memory found this set of tasks to be more challenging than those modified with temporal delay, it is no surprise that agents learning to use bit memory also struggled to perform well. Results for agents that learned to use bit memory without eligibility traces demonstrated poor performance; for two or more dependent decisions, mean agent performances were indistinguishable from each other and were asymptotic at roughly -0.2 reward per episode.

Figure 7.5 shows the results for agents that learned to use bit memory with eligibility traces in TMaze tasks with one to eight dependent decisions and an eligibility trace decay rate of 0.7. Mean learning performance in tasks with two dependent decisions quickly converged to near optimal reward per episode, and quickly decreased from there.

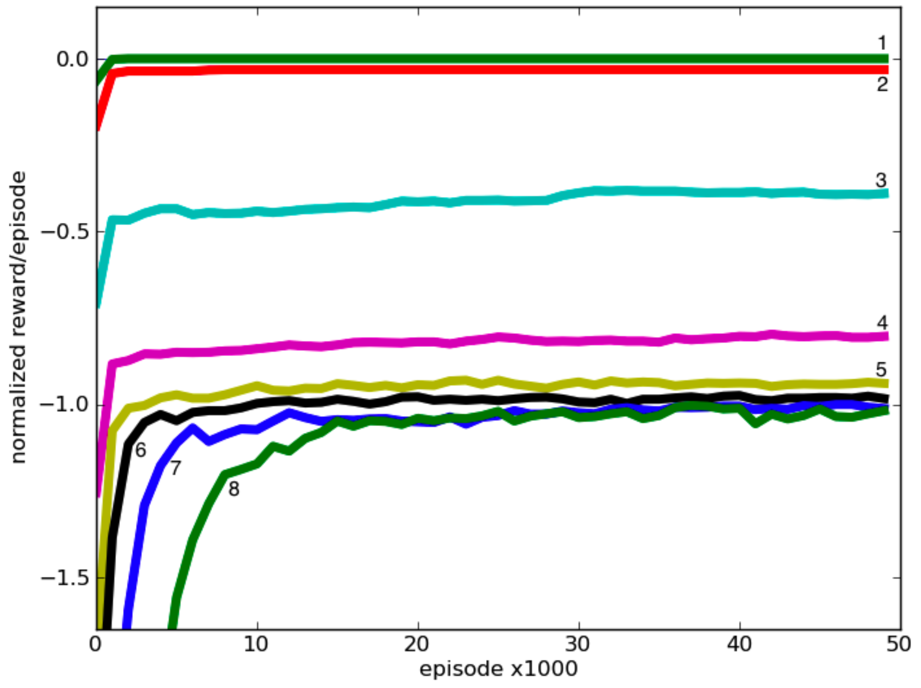

Figure 7.5: Agent performance when learning to use bit memory with eligibility traces in the TMaze task extended with additional dependent actions and unique observations, for one to eight dependent actions and an eligibility trace decay rate of 0.7.

7.2.2 Dependent Decisions with Identical Observations and Agents with Gated Memory

In the identical case, the observations made at each location corresponding to a dependent decision (e.g. the nth dependent decision) are all aliased with each other. In TMazes extended along the temporal delay dimension, learning performance degraded when moving from the unique case to the identical case. Therefore we predicted that we would observe a similar phenomena when moving to the identical observation case in TMazes extended with additional dependent decisions.

Figure 7.6 shows the mean learning curves for agents that learned to use gated memory in TMaze tasks extended with additional dependent actions and identical observations. In the base TMaze (with one dependent action), agents converge to the optimal policy very quickly (as we have seen in previous sections). However, for two or more dependent actions with identical observations, agents converge to the same suboptimal behavior and are incapable of learning to perform the task.

In the previous section, we observed that the convergence time of gated memory scaled exponentially with the number of dependent decisions. By moving from unique observations to identical observations, we expected performance to degrade. Given that the upper bound on convergence time was exponential scaling, it is unsurprising that agents were unable to learn to use memory in this task without eligibility traces.

Figure 7.7 shows the results for agents that learn to use gated memory in the same set of tasks but with eligibility traces. Here we observe that agents can learn to use memory and converge to near-optimal behaviors in tasks with one to four dependent decisions, but that performance begins to degrade noticeably for five and more dependent decisions.

These results are the first that we have seen in which an agent learning to use gated memory cannot converge to a near-optimal behavior either with or without eligibility traces.

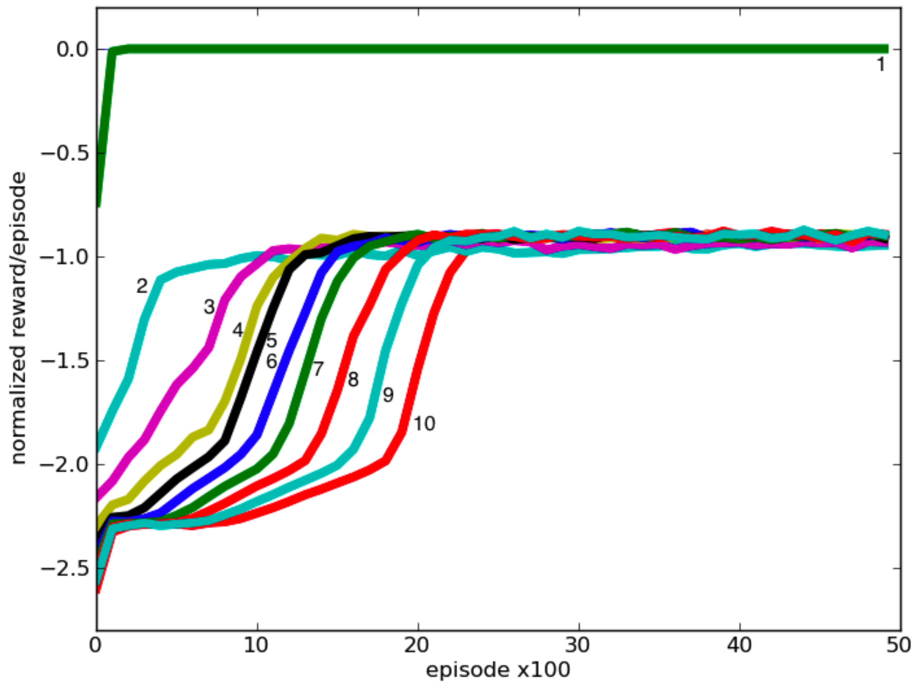

Figure 7.6: Agent performance when learning to use gated memory in the TMaze task extended with additional dependent actions and identical observations, for one to ten dependent actions.

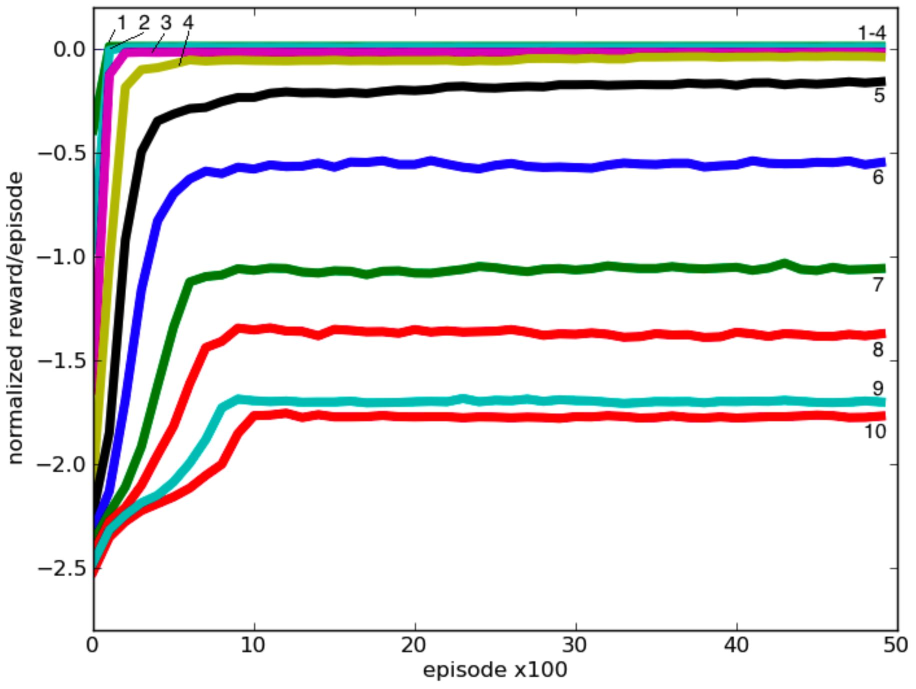

Figure 7.7: Agent performance when learning to use gated memory with eligibility traces in the TMaze task extended with additional dependent actions and identical observations, for one to ten dependent actions and an eligibility trace decay rate of 0.7.

The state diagram for an agent using gated memory in the TMaze task with two dependent actions and identical observations is shown in Figure 7.8.

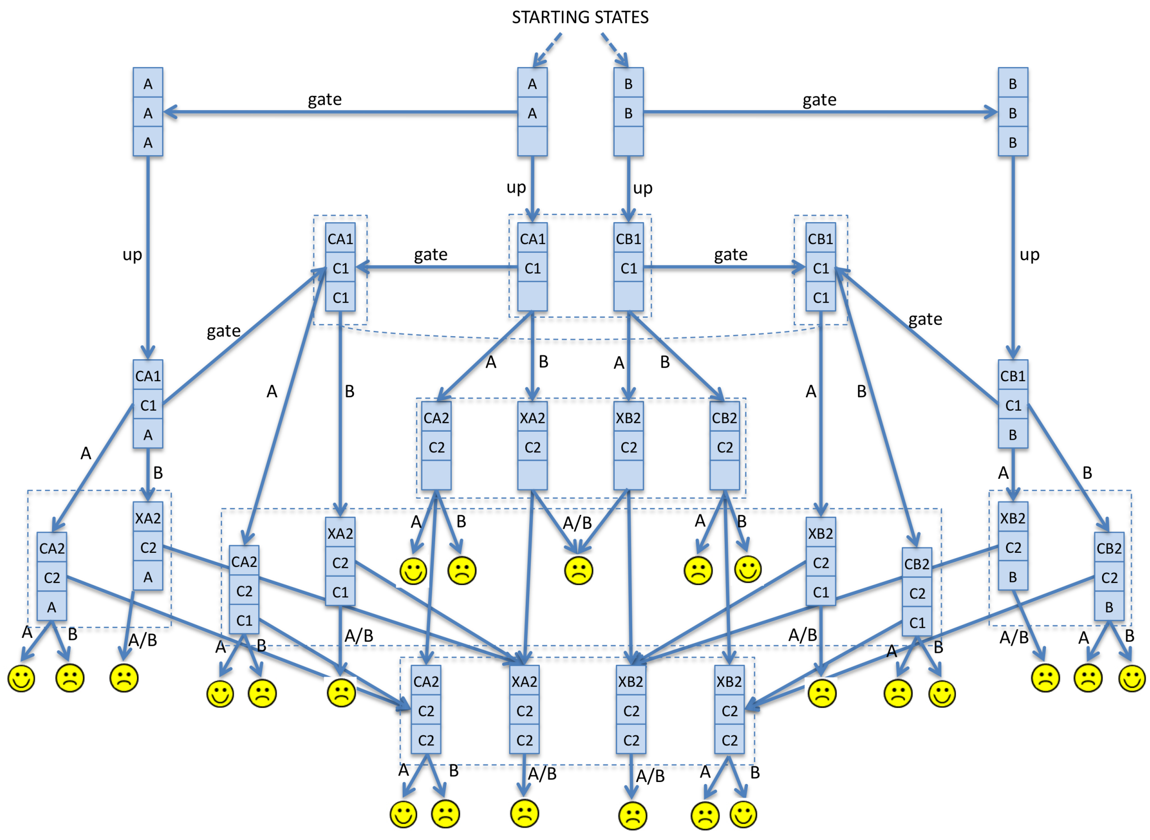

Figure 7.8: State diagram for an agent endowed with gated memory performing the TMaze task with two dependent choices and identical observations.

There are several differences between this state diagram analysis and the state diagram for the unique observation case (above, Figure 7.4). The most important difference is that in the unique case the optimal behavior did not traverse any aliased states. In the aliased case, however, the optimal behavior traverses a pair of aliased states in either case (when an agent's internal memory contains knowledge of the salient observation and the current observation is C_2).

This turns out to be a significant barrier to learning the optimal policy, but yet is also different from the aliased states when an agent is learning to use bit memory.

When an agent learns to use bit memory (in the base TMaze, as in Figure 5.2) all states are reachable, regardless of the starting state and the identity of the salient observation. An agent using bit memory then had to simultaneously learn both a stable behavior over memory as well as a stable behavior in the environment, both of which depended on each other.

In the case of an agent learning to use gated memory in this task, an agent faces the same problem but not all states are reachable from each other; in fact, the state space is essentially split down the middle, even though some states are aliased with each other. Although the optimal behavior must traverse a pair of aliased states, if an agent could learn to always select action A when its internal state were (C_2, A) (and the equivalent for the other salient symbol) then it could perform optimally.

7.2.2.1 Constructing Representative POMDPs

As discussed above, we believe that the difference in the asymptotic mean performances of agents learning to use gated memory in TMaze tasks with additional dependent decisions and identical observations can be explained by the optimal behavior in the task traversing aliased states, while in the base TMaze task the optimal behavior traverses only completely observable states.

The portion of the state space of interest is shown in Figure 7.9. If an agent gates internal memory, and then moves up in the environment, it will have three possible actions that it can select: A, B, and to gate internal memory. If it gates internal memory, it will enter an aliased state and remain in aliased states until it selects a terminal action. At this point, the best that it can hope to do is guess as to which terminal action is correct and receive, on average, 0.0 terminal reward (less penalties for each action that it selects).

If instead of the gate action the agent selects action B, the agent is guaranteed to receive -1.0 terminal reward either when it selects actions A or B immediately, or

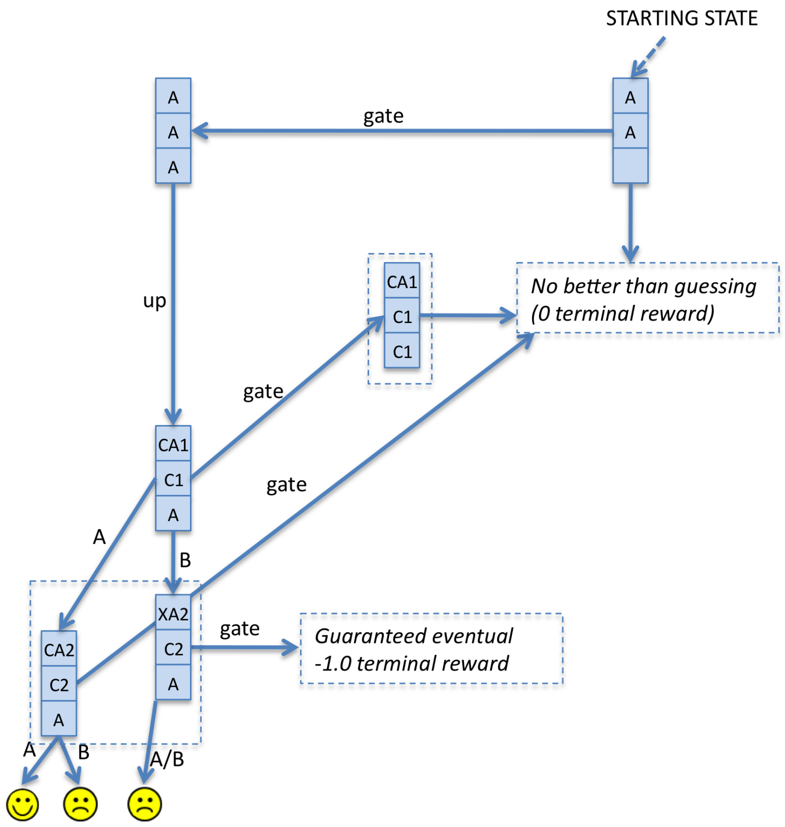

Figure 7.9: The portion of the state diagram that is of particular interest for an agent endowed with gated memory performing in a TMaze task with two dependent decisions and identical observations.

after gating memory and arriving in a different aliased state.

Finally, if instead of gate or B the agent selects A, then selecting A again will lead to 1.0 terminal reward and B will result in -1.0 terminal reward. Given this, it might seem reasonable that an online temporal difference agent could learn the optimal behavior in this subset of the state diagram: given sufficient exploration, the action A in the aliased state space of interest would result in higher expected reward than either selecting action B or gating memory, and would lead to the agent eventually selecting action A twice in a row and learning the optimal behavior.

We constructed two small POMDPs that represent an abstraction of the state space of interest in order to isolate the task characteristics that were responsible for agents' difficulty in learning to use memory. Figure 7.10 shows the first of two POMDPs: a single state (state 0) with two actions, each of which transitions deterministically to two different states, states 1 and 2. An agent observes A from state 0 and B in both states 1 and 2. Rewards for terminal actions from states 1 and 2 are as shown in the figure; the optimal behavior is for an agent to select the left actions twice in a row, which leads to reward of 1; all other behaviors result in reward of -1.

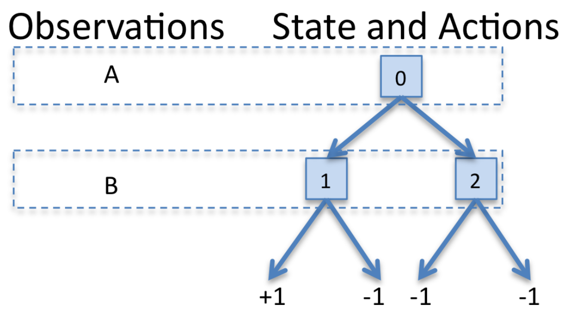

Figure 7.10: A POMDP that captures aspects of the state diagram for an agent with gated memory in TMazes extended with additional decisions and identical observations.

The second POMDP is a small extension of the first one. As illustrated in Figure 7.11, an additional terminal action is added in state 0 that results in 0 reward. The optimal behavior in the task remains the same.

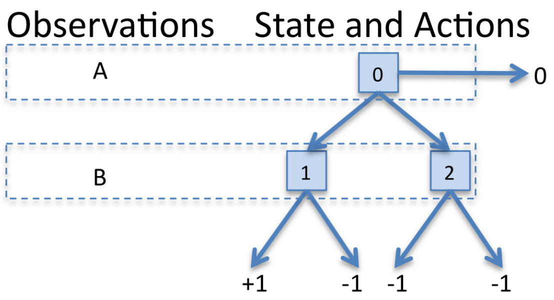

Figure 7.11: POMDP from Figure 7.10 extended with an additional action in state 0 which results in terminal reward of 0.0.

We collected results in four experimental conditions: agents in each of the two POMDP tasks described above, and with and without eligibility traces. The mean performances of agents under each condition are shown in Figure 7.12. From the figure, it is clear that agents perform significantly better in the first task than in the second, and that the third action available in state 0 dramatically affects the behaviors that agents can converge to.

When agents learn with eligibility traces in the first task, they quickly converge to the optimal behavior. Without eligibility traces, performance slowly converges to a sub-optimal but near-optimal asymptote.

When agents learn in the second task, performance is noticeably worse. Agents with eligibility traces receive a mean of 0.4 reward per episode, or the same as selecting the 0.0 terminal reward action half of the time and following the optimal behavior the other half. Without eligibility traces, agents in the second task learn only to select the terminal reward in the starting state, resulting in performance of 0.0 reward per episode.

The results from these experiments with simple POMDPs demonstrate an interesting phenomena: that by adding a single action from a state with sub-optimal expected reward, an agent can be distracted from learning an optimal policy in a partially observable task. While agents in the first task performed either optimally or

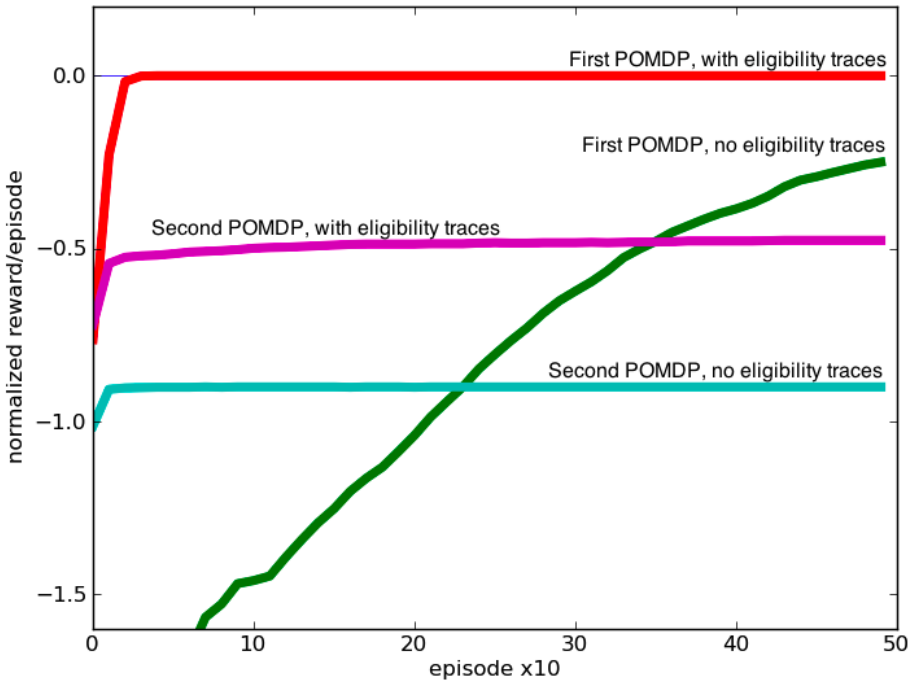

Figure 7.12: Mean agent performances in both POMDP tasks, with and without eligibility traces.

relatively well in the face of partial observability, agents in the second modified task fared poorly overall and instead of finding the optimal policy in the face of partial observability instead selected the terminal action with lower overall reward.

This aids us in understanding the results that we observed when modifying TMaze tasks with additional decisions and identical observations in conjunction with agents learning to use gated memory. Although learning the optimal action in that setting seems like it is a reasonable thing for an online temporal difference agent to learn, it in fact is difficult in part because of the rest of the state action space of the task. While looking at trajectories through the state space is useful and informative, other actions and trajectories cannot be ignored.

7.2.3 Dependent Decisions with Identical Observations and Agents with Bit Memory

Given that agents failed to learn to use gated memory to perform the tasks, it is unsurprising that agents learning to use bit memory exhibit similarly poor behaviors. Figure 7.13 shows results for agents that learned to use bit memory with eligibility traces in the TMaze tasks extended with additional dependent actions. For two or more dependent actions, agents converge to considerably less than optimal behaviors.

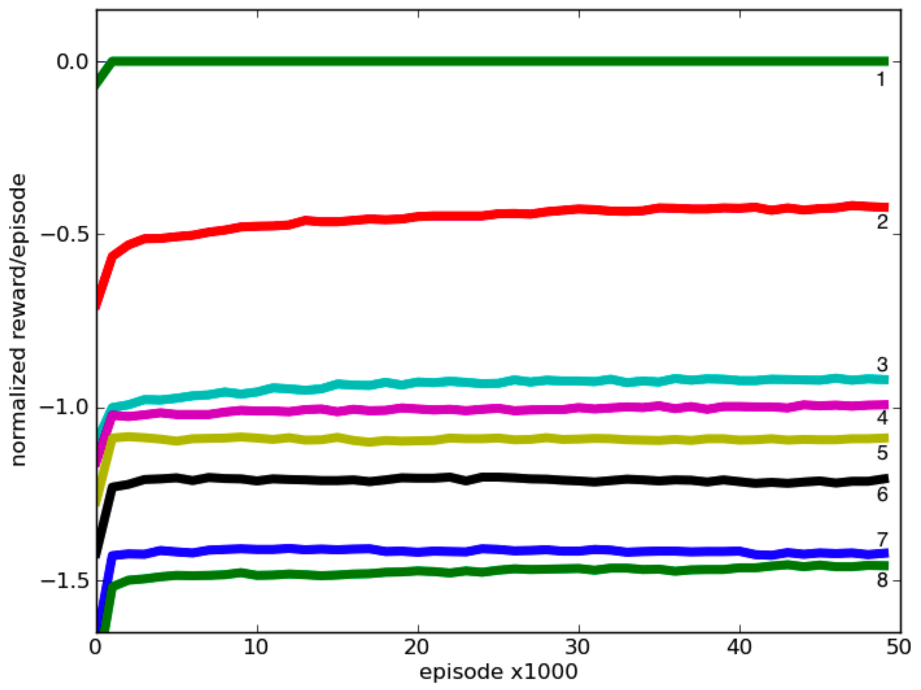

Figure 7.13: Agent performance when learning to use bit memory with eligibility traces in the TMaze task extended with additional dependent actions and identical observations, for one to eight dependent actions and an eligibility trace decay rate of 0.7.

CHAPTER VIII

Number of Distinct Symbols

8.1 Description of Number of Distinct Symbols

In the base TMaze task (shown in Figure 3.5), the agent observes one of two salient symbols and must maintain knowledge of the salient observation in order to correctly perform the task. The TMaze task can be extended by supplementing the alphabet of salient symbols that can be observed with new, additional symbols in the initial location.

Figure 8.1 shows a TMaze extended with additional distinct symbols. Rather than two possible symbols in the initial location, this TMaze has five symbols. Each symbol is correlated with a correct action in the choice location, which results in positive reward when it is selected after having observed that particular symbol; the other possible action results in negative reward, as in the base TMaze task.

In order to successfully perform the TMaze task extended with additional distinct symbols, an agent must:

1. Encode and store knowledge of the symbol presented in the first location.

2. Move to the choice location.

3. Not modify the contents of internal working memory before the next step.

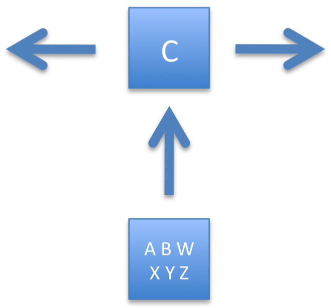

Figure 8.1: The TMaze task extended along the dimension of the amount of salient knowledge

4. Associate the salient knowledge with the correct action and select that action in the environment.

As the number of distinct symbols increase, the difficulty of the task increases, but more superficially than in the other parameterizations that we have seen. Whereas increasing temporal delay and the number of dependent actions has a relationship with how aliased the task is, increasing the amount of distinct symbols only increases the space of the possible observations that an agent can make, and does not affect the degree to which task states are aliased.

As the number of distinct symbols increases, the representational capacity of memory must also increase. For example, if there are three distinct symbols and three distinct corresponding dependent actions, an agent's memory model must be capable of representing three distinct values in order for the agent to exhibit the optimal behavior in the task. In such an example, a bit memory representing only binary values would be incapable of supporting the optimal task behavior.

8.2 Number of Distinct Symbols Experiments

We explored varying the number of distinct symbols in order to determine how creating a larger vocabulary of salient knowledge would affect an agent's ability to learn to perform the TMaze task. While both previous sets of experiments involved different variations on whether observations were identical or unique (which affected which environmental states were aliased), our investigation in this section varies the number of distinct symbols by parameterizing tasks in only a single dimension.

We observed the performance of agents that learned to use gated memory in TMaze tasks extended with varying numbers of distinct symbols and corresponding dependent actions. Figure 8.2 shows the mean performances of agents for TMaze tasks with thirty to two hundred distinct symbols and corresponding dependent actions. All agents that we observed converged to optimal behavior.

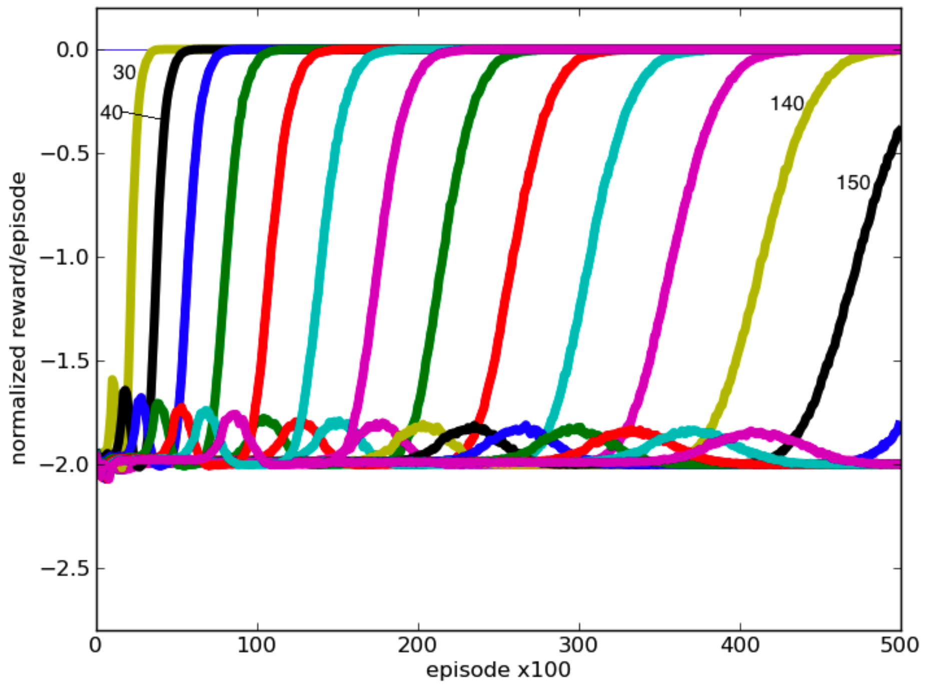

Figure 8.2: Agent performance when learning to use gated memory in TMaze tasks with varying numbers of distinct symbols, for thirty to two hundred distinct symbols and corresponding actions.

We plotted the number of episodes to convergence to optimal behavior for agents in the same conditions as above, but for two to one hundred forty distinct symbols, in Figure 8.3. Using quadratic regression we fit a curve to the data, which resulted in an R^2 fit of 0.9999; thus the amount of experience required for an agent to converge to the optimal behavior is quadratic in the number of distinct symbols in the task.

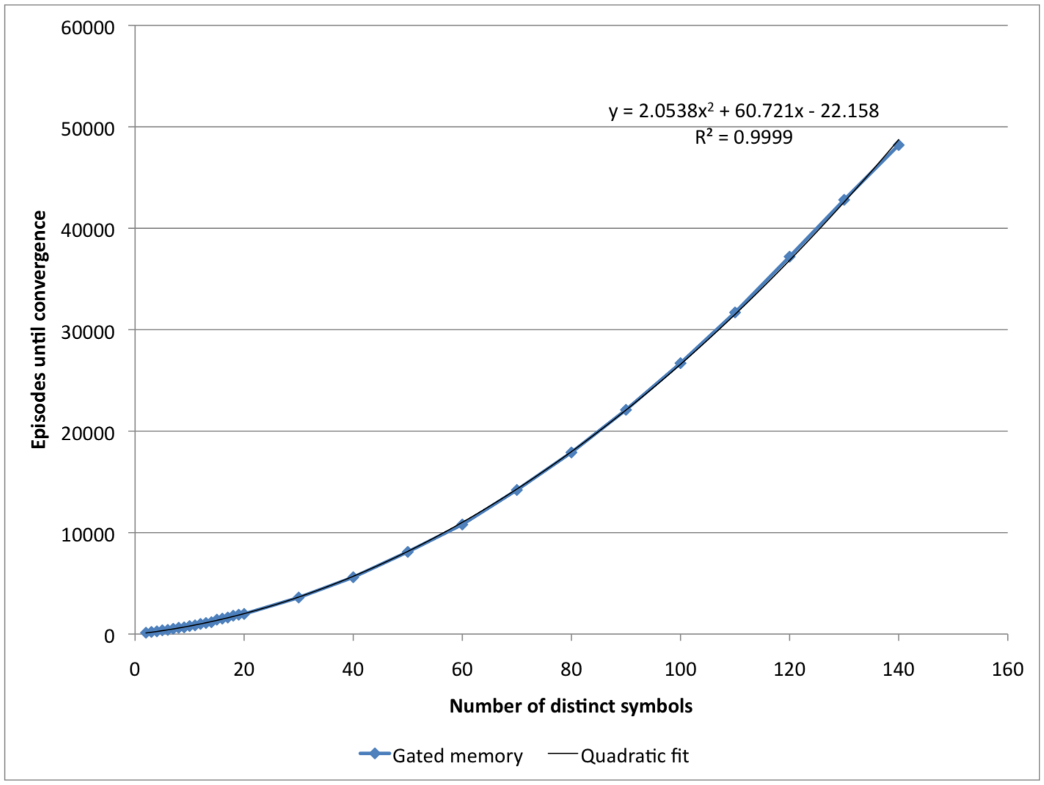

Figure 8.3: Amount of experience required for agents learning to use gated memory in TMaze tasks with various numbers of distinct symbols to converge to optimal behavior, for two to one hundred forty distinct symbols and corresponding actions.

This is a sensible result. Consider the state diagram of an agent learning to use gated memory in the base TMaze task (Figure 5.2). In the state diagram for the base task, there are two trajectories that cover the optimal behavior and in which states are not aliased with any others. As the number of distinct symbols increases, so too do the number of unaliased trajectories that lead to optimal behavior (each corresponding to a unique starting state and distinct symbol).

However, the number of distinct aliased states, or the cardinality of the set of aliased states, does not increase. The expected reward of any terminal action selected in the set of aliased states decreases with the number of distinct symbols, where the expected reward is $1(1/n) - 1(n - 1/n)$, or $2 - n/n$. Thus, as the number of distinct symbols increases the perceived value of aliased states becomes worse and worse, and thus agents have more and more motivation to explore the trajectories that are completely observable and lead to expected reward.

We also explored agents that learned to use bit memory in tasks with varying numbers of distinct symbols. As agents that learn to use bit memory without eligibility traces fare poorly in the base TMaze task, we only explored agents that learned to use bit memory in these tasks in conjunction with eligibility traces; specifically, the optimal eligibility trace decay rate for these tasks was found to be 0.9. As binary bit memory is insufficient to support optimal behavior in tasks extended with more than two distinct salient symbols, we used n-ary bit memories with set actions (instead of toggle actions), where n is the number of distinct symbols.

The results of agents that learned to use bit memory with eligibility traces in TMazes with two to eight distinct symbols (and corresponding dependent actions) are shown in Figure 8.4. For each additional distinct symbol beyond the initial two, the agent's asymptotic performance decreases slightly but noticeably.

Initially, we predicted that an individual behavior analysis of these results would show patterns of behavior similar to those that we have already seen: as the number of distinct symbols increase, the ratio of agents converging to optimal behavior to agents converging to sub-optimal behaviors would decrease. Instead, we found that nearly all agents, for all conditions, converged to behaviors in which they always selected the correct dependent action and received positive terminal reward.

The reason that asymptotic performance decreases as the number of distinct symbols increases is that, when presented with a salient symbol, agents set bit memory several times in succession before settling on a value of memory and moving forward in the task (and then selecting the correct terminal action). For example, when ob-

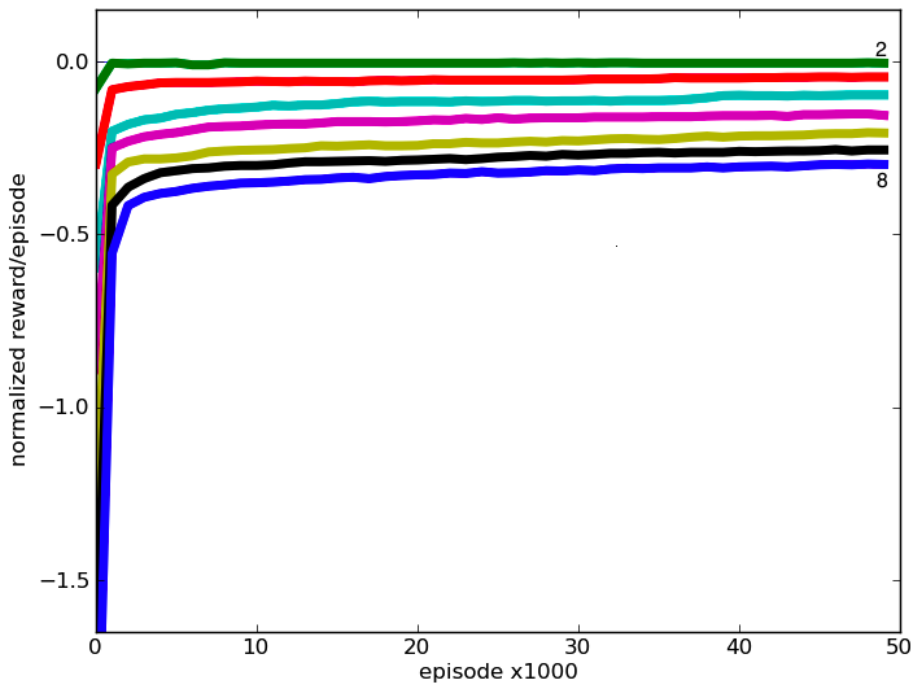

Figure 8.4: Agent performance when learning to use bit memory with eligibility traces in TMaze tasks with varying numbers of distinct symbols, for two to eight distinct symbols and corresponding actions.

serving salient symbol $A3$, an agent might set the contents of memory to 5, then set the contents of memory to 6, then move forward and select terminal action $A3$. In some situations the same agent might set memory contents once and immediately move forward, in others the agent might set action more than twice before moving forward.

As the agent is penalized -0.1 reward for each selected action, these additional internal actions prevent the agent from exhibiting the true optimal behavior. When using conventional binary bit memory we don't observe similar behaviors because the representational capacity of memory is limited. If the agent were penalized less for using internal memory than for actions in the environment (for example, if penalties were proportional to the time it took to execute internal and external actions), then the asymptotic behaviors exhibited by the agents would be relatively improved.

This is the first dimension of task for which agents using bit memory has converged to behaviors in which it always selected the correct terminal action in the environment.

CHAPTER IX

Concurrent Knowledge

9.1 Description of Concurrent Knowledge

All of the TMaze extensions that we have discussed to this point have involved a single salient symbol that is observed. Instead of representing knowledge as a single symbol internally, we can use an alternate representation in which both observations and knowledge in memory are sets of symbols, rather than single symbols.

Agents endowed with bit memory are not directly affected by this alternate representation, but agents endowed with gated memory can be if the internal actions available to them do not allow for the entire set of observed symbols to be gated to memory at once, but instead allow symbols to be gated individually. If observations can be gated as a set, rather than individually, then the task is reduced to that of varying the number of distinct symbols as above; explorations in this dimension will exclusively be concerned with individual actions over slots of memory.

Figure 9.1 illustrates a TMaze extended along the concurrent knowledge dimension with two concurrent salient observations. When the agent is in the initial location, it observes a vector of two symbols: one symbol of A, B and one symbol of X, Y, for four possible combinations.

Necessarily there are four dependent actions, one for each unique combination of salient observations. If the number of dependent actions available in the choice

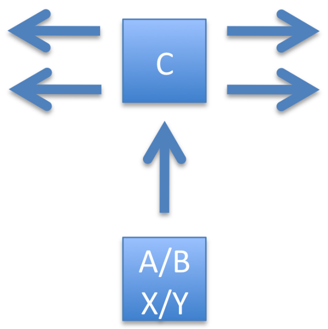

Figure 9.1: The TMaze task extended with concurrent knowledge.

location was less than the number of unique combinations of salient symbols, then the agent might not need to gate all perceptual symbols to memory in order to select the correct dependent action in the choice location.

This task is similar to the tasks in Gorski & Laird (2009, 2011) in which an agent endowed with episodic memory learned to construct an episodic memory retrieval cue. In that work, an agent learned which features were salient to the episodic memory retrieval; in this work, *all* features are salient and the agent must learn to encode all of them to working memory.

In order to maintain multiple symbols in memory, an agent's internal memory must have additional slots. Thus to completely represent n unique perceptual symbols, a gated memory must also have n slots. Bit memory must have the same representational capacity as the number of unique combinations of perceptual symbols; for example, if there are eight possible combinations of symbols then binary bit memory would require three slots, ternary bit memory would require two slots, or a single slot of 8-ary bit memory would also be sufficient.

In order to successfully perform the TMaze task extended with concurrent knowledge, an agent must:

1. Encode and store knowledge of the combination of salient symbols observed in the initial location. For an agent using gated memory, this involves gating

each symbol to memory independently and without interfering with previously gated symbols. For an agent using bit memory, it means encoding each unique combination of observed symbols as a unique combination of bits in memory.

2. Move to the choice location.

3. Not modify the contents of internal working memory before the next step.

4. Associate the salient knowledge with the correct action and select that action in the environment.

As the number of concurrent salient symbols increases, so too does the difficulty of the task. As was the case with additional distinct symbols, additional concurrent symbols do not affect the degree of aliasing within the task. However additional concurrent symbols do affect the number of environmental actions that an agent must take in the task, the size of the observation space, and the number of actions available in the choice location.

9.2 Concurrent Knowledge Experiments

Extending the TMaze task with concurrent knowledge involves adding additional salient symbols to the starting location, as well as additional dependent actions to the choice location, where there is one terminal dependent action per unique combination of salient symbols. For example, when there are three binary salient symbols then there are nine dependent actions.

Agents that learn to use gated memory in TMazes with concurrent salient knowledge and without eligibility traces converge to the optimal behaviors, but the agents require significant amounts of experience relative to previous experiments. We present plots of agents learning with eligibility traces only because they require less experience and the mean learning curves are qualitatively similar with respect to how learning performances decrease in quality with the amount of concurrent knowledge.

The mean performances for agents learning to use gated memory in TMazes with concurrent salient knowledge are illustrated in Figure 9.2, for one to eight concurrent salient symbols. Agents that learn to perform tasks for one to three concurrent salient symbols converge to the optimal behavior very quickly, but time to convergence scales very poorly for agents in tasks with more concurrent symbols. Although not all learning curves converge to optimal behavior in the figure, the mean performances do converge to optimal in all eight cases.

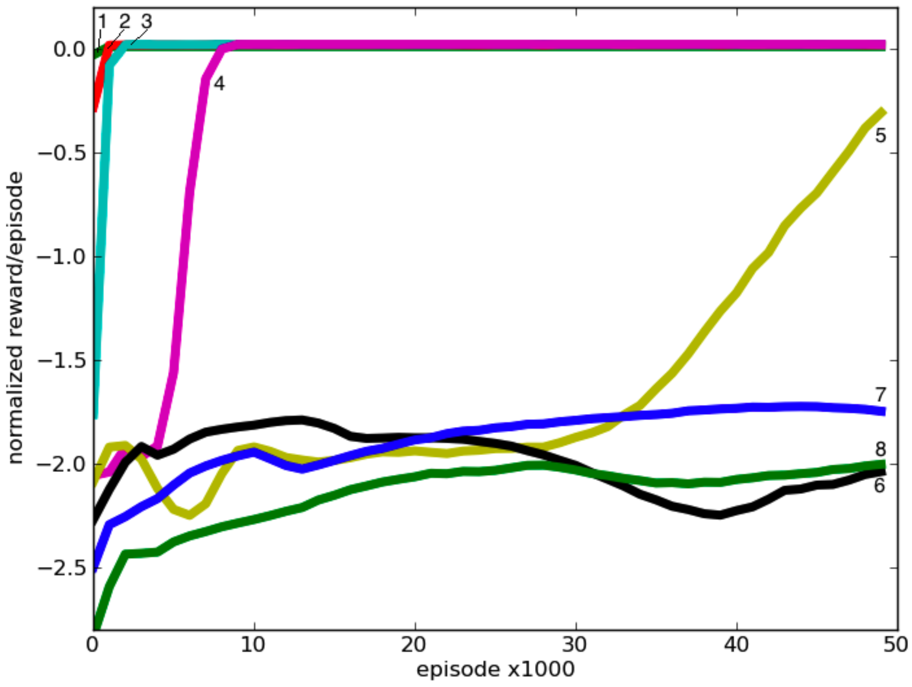

Figure 9.2: Mean agent performance when learning to use gated memory with eligibility traces in TMaze tasks with concurrent knowledge, for one to eight concurrent salient symbols.

Given these results, the most obvious question is: why does the time to convergence increase so dramatically with the amount of concurrent knowledge? The answer is simply that the size of the state space of the task grows exponentially with the amount of concurrent knowledge and the number of gated memory slots. The number of possible internal states in the starting location is $(3^n + 2^n)$ alone. As additional concurrent knowledge is added to the task, there are still completely observable (unaliased) trajectories through the state space of the task and thus the agent can, with sufficient experience, determine how to act. There are simply many more task states that the agent can experience when a task is extended with concurrent knowledge relative to previous dimensions that we have explored, requiring significantly more experience in order for agents to learn to perform tasks extended along this dimension.

As agents required exponentially more experience in order to converge to optimal behaviors when learning to use gated memory, we expected agents that learn to use bit memory to fare poorly in this domain.

The mean performances for agents learning to use bit memory with eligibility traces in TMaze tasks with one to four concurrent salient symbols are illustrated in Figure 9.3. For each salient symbol, agents were endowed with a corresponding slot of bit memory; for example, in a TMaze task with four concurrent salient symbols, agents learned to use a four-slot bit memory (where each slot contained a binary value that was controlled with set actions). Although agents only converge to the optimal behavior in the base task, agent performance scales relatively well as the number of concurrent symbols increases, particularly given how poorly performances scaled for agents learning to use gated memory in the same tasks.

Overall these performances were significantly better than what we expected to observe, relative to how bit memory has fared when supporting behavior in challenging tasks that we have already observed. Note that agents converge to stable, though often sub-optimal, behaviors with about the same amount of experience as for agents using gated memory in the same task conditions.

This is the first experiment in which we explore agents that learn to use more than

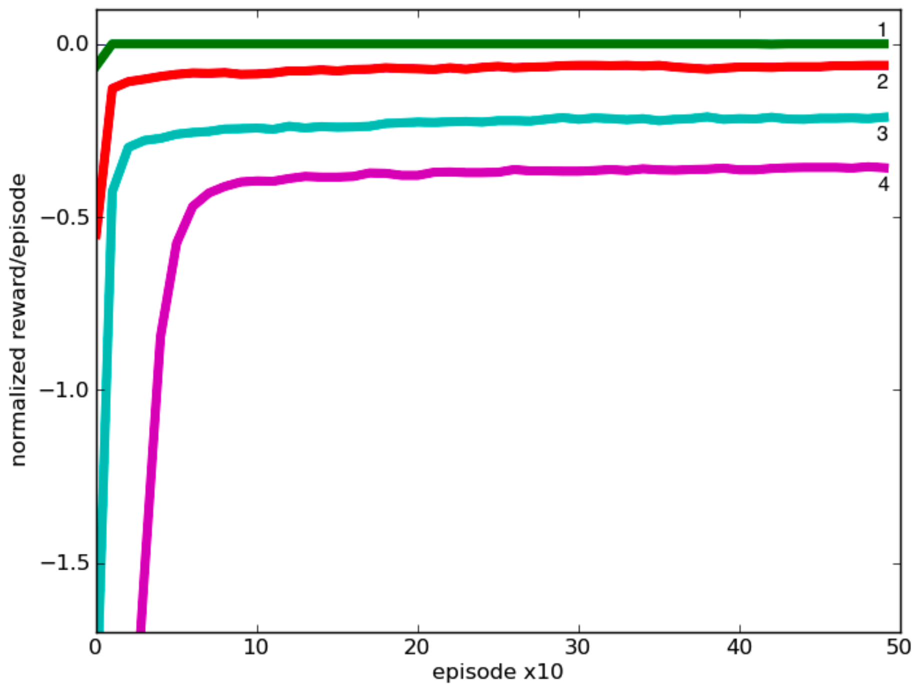

Figure 9.3: Mean agent performances when learning to use bit memory with eligibility traces in TMaze tasks with concurrent knowledge, for one to four concurrent salient symbols.

a single slot of bit memory. In order to better understand the effects of learning to use multiple bits of memory, we ran a control experiment in which agents were endowed with one to four slots of binary bit memory, but learned to perform in the base TMaze task. The mean performances of agent performances are shown in Figure 9.4.

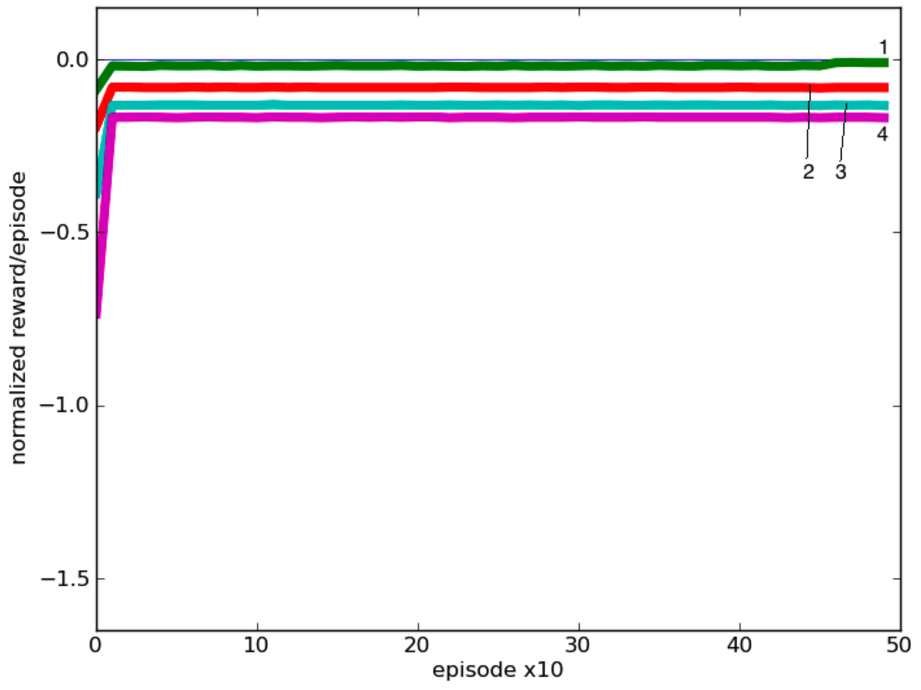

Figure 9.4: Mean agent performances when learning to use bit memory with eligibility traces in the base TMaze task, for one to four slots of bit memory.

As seen in the figure, agents converge to behaviors that are slightly sub-optimal when endowed with additional slots of bit memory, and the more slots they are endowed with the worse their performances. An individual analysis showed that this was because agents would often set more than a single slot of memory before advancing in the task. This behavior is similar to the behaviors observed in agents performing in tasks with additional distinct symbols (Section 8.2), where agents selected consecutive internal actions before arriving at a memory state in which they advanced to the choice location.

In this case, agents lacked the capacity to generalize and learned optimal behaviors in a subset of the internal state space. Then for the remaining internal states they learned to configure memory in such a way that they would know how to act, selecting unnecessary internal actions in order to receive a positive terminal reward.

From the results of this experiment, we can conclude that although having additional slots of bit memory does cause some slight negative impact to the convergent behaviors that agents exhibit, it does not explain why agents could not converge to the optimal behavior in tasks with concurrent knowledge. We thus conducted an individual behavior analysis so that we could characterize the behaviors that agents were exhibiting after convergence.

We observed two phenomena when conducting the individual behavior analysis that explains why agents do not converge to the optimal behaviors. First, agents would occasionally set the same slot of memory twice. For example, in one agent that we observed learning to perform a TMaze task with three concurrent symbols, the trace of actions that it selected in the starting location was: [$\text{slot}_1 = 1$, $\text{slot}_2 = 0$, $\text{slot}_1 = 0$, $\text{slot}_0 = 0$, $\text{slot}_2 = 1$, up, terminal_4]. In that episode, the agent selected the correct terminal action and received positive reward, but selected five internal actions when it only had three slots of internal memory. The agent thus had learned to select the correct terminal action, but was taking more actions than necessary when setting internal memory in the starting location. Had the agent explored more of the task space, it would have been able to perform better asymptotically.

The second phenomena that we observed was that agents would sometimes not select the correct terminal actions. For example, in the same agent trace as in the previous paragraph, in a later episode the agent observed three salient symbols in the starting location: [A, D, F], immediately moved forward in the task, and selected action terminal_1, which was the incorrect terminal action. Although the agent learned to correctly use memory in most of the task state space, there were some starting states in which it had learned to not use memory and simply move forward and receive negative reward when it selected the wrong terminal action.

Thus, while agents learned to use bit memory fairly well and in the mean received near-optimal reward per episode, agents didn't always use memory optimally and didn't always learn to perform the complete task.

CHAPTER X

Second-Order Knowledge

10.1 Description of Second-Order Knowledge

In all of the TMaze extensions that we have discussed earlier, the correct dependent action in the choice location depends directly on the salient knowledge that was observed. For example, in the base TMaze if the agent observes symbol A then, across all episode instances, the identity of the correct action to select in the choice location remains the same.

The TMaze task can be modified so that the correct action depends not on the symbol observed in the initial location, but depends on the relationship between that symbol and the symbol observed in the choice location. In addition to modifying the dynamics of the TMaze task, this requires modifying the agent framework to support additional internal actions such as equality and inequality tests to compare memory contents to the current observation.

This task is analogous to an agent being told the name of a street in the initial location, and then when it reaches the choice location turning to go down the street that was named earlier – but it selects the street to turn down based on the equality relationships of its options to the name it has stored in memory, rather than simply turning left or right based on the knowledge it has in memory.

Figure 10.1 illustrates a TMaze task that has been modified to condition the correct action in the choice location on second-order knowledge of the relationship between the salient observation and an observation made in the choice location. In the second-order task, there are two actions in the choice location: one that is correct if the observations in the initial and choice locations are identical, and one that is correct if they are different.

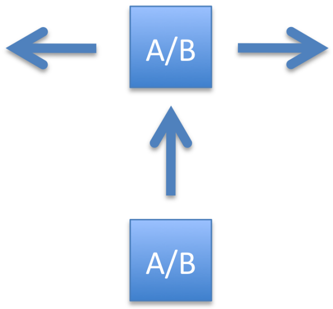

Figure 10.1: The TMaze task extended with second-order knowledge.

This modified TMaze task can be parameterized by varying the amount of salient knowledge.

As mentioned above, the agent framework must also be modified to support the correct execution of this task. An agent using a memory that encodes and stores concrete knowledge, as with gated memory, must be endowed with internal actions that can compare the currently maintained knowledge with an agents current observation, placing the result of the internal comparison operation into the agent's internal state representation. This new internal state representation would thus contain the contents of internal memory and the immediate perceived observation symbol, as well as the results of the last internal comparison action.

An agent using a memory that does stores abstract knowledge, like bit memory, cannot support second-order comparison operations as there are no perceptually grounded symbols in memory to compare directly to current observations.

In order to successfully perform the modified TMaze task with second-order knowledge, an agent may (not must, as detailed below):

1. Encode and store knowledge of the symbol presented in the first location, either A or B, using a memory that stores concrete knowledge.
2. Move to the choice location.
3. Not modify the contents of internal working memory before the next step.
4. Perform an internal action comparing the encoded salient knowledge with the current observation.
5. Associate the results of the internal action with the correct environment action and select it.

Changing the agent's internal state representation in this manner has two effects. First, it allows an agent to use second-order knowledge to perform the task, rather than learning a reactive policy that avoids comparing memory contents to perception. Second, it affords an agent with some capacity for generalization: rather than learning a behavior that is dependent on all possible combinations of symbols in memory and perception, an agent's behavior could depend on the relationship between symbols instead, if it had the ability to generalize over internal agent states (which our framework does not support).

Note that if an agent's internal state contains both the contents of internal memory and current observation, the agent can use that knowledge in order to determine how to correctly act in the world without relying on second-order knowledge. If the agent learns a behavior that does not involve performing an internal comparison action, then the agent would successfully perform the task by:

1. Encode and store knowledge of the symbol presented in the first location, either A or B, using a memory that stores concrete knowledge.

2. Move to the choice location.

3. Not modify the contents of internal working memory before the next step.

4. Select the terminal environment action that is associated with the combination of memory contents and immediate observation, but without having explicitly tested the equality relationship between memory contents and immediate observation with its available internal action.

10.2 Summary of Experiments

The results for agents that learned to use gated memory in TMaze tasks extended with second-order knowledge confirm our prediction from Section 10.1, where we observed that agents could learn to perform these tasks without selecting the internal comparison actions. Indeed, agents learning to use gated memory without eligibility traces quickly converged to the optimal behavior in tasks for two to ten distinct salient symbols and terminal actions that depended on the second-order equality relationships between the salient symbol and the symbol observed in the choice location.

We performed an individual behavior analysis on agents learning to perform in second-order TMaze tasks with two, six, and ten distinct salient symbols, and in all cases agents did not learn to perform the internal comparison operations but simply learned the behavior of selecting a terminal action in the choice location depending on the agent's internal state without having performed the comparison operation. Agents learned this behavior because it is more rewarding: the selection of an additional internal comparison action results in an additional -0.1 reward being received.

In order to confirm that agents could learn to use second-order knowledge, we modified the TMaze task and agent framework to prevent the terminal task actions from being available until after the agent selected an internal comparison action. In this case, agents converged to optimal behaviors and successfully learned to use the second-order knowledge in memory. Thus, while agents can learn to use second-order knowledge, they will likely learn a behavior that doesn't rely on second-order internal actions if that behavior results in higher reward.

These results raise the issue of generalization in reinforcement learning agents. In the reinforcement learning paradigm, an agent's goal is to optimize its behavior for a particular objective function. Agents are not encouraged to generalize their knowledge representations, as learning a general representation or spending time in the current task preparing for future tasks are orthogonal to the task at hand: balancing immediate exploration and exploitation. While agents did not learn to use memory to support cognitive capabilities involving second-order knowledge in our TMaze experiments, it does not imply that agents could not learn to use memory actions relating to second-order knowledge, simply that conventional reinforcement learning tasks combined with working memory place environmental constraints on an agent that encourage it to not learn those immediate behaviors.

We experimented with agents learning to use gated memory exclusively because, as we noted in Section 10.1, agents using bit memory are unable to support second-order comparison operations between the contents of memory and the current perceptual observation as they store abstract knowledge that is not perceptually grounded, and therefore the results of such a comparison are undefined.

CHAPTER XI

Discussion

In this chapter we discuss trends and conclusions from across dimensions of tasks. After summarizing the results of our experiments and discussing classes of behavior that were observed, we continue by contrasting the performances of agents using gated and bit memory models. We then briefly touch on issues such as evaluating learning performance, when eligibility traces are appropriate and when they are not, parameter selection, and long lived agents. We conclude the chapter by reiterating conclusions drawn in this thesis.

11.1 Discussion of Results Along Dimensions of Task

Given that we presented results for many different experiments in Chapter IV, it is helpful to summarize the combinations of memory model, learning algorithm and task dimension that we included in this thesis. Table 11.1 summarizes the experiments that we presented and shows the coverage of experiments that were included in this thesis. The left column in the table lists groups of tasks that we explored in our experiments, while the other two columns summarize which combinations of memory and learning algorithm we explored. Q(0) refers to Q-learning with pure temporal difference updates, while Q(λ) refers to Q-learning with eligibility traces and the best performing decay rate as found via parameter sweeps.

Table 11.1: Summary of experiments presented in Chapter IV.

	Gated		**Bit**	
	Q(0)	Q(λ)	Q(0)	Q(λ)
Base TMaze	✓	✓	✓	✓
Architectural modifications				
Prohibit internal			✓	
Null initialized			✓	
Temporal Delay				
Unique	✓			✓
Identical	✓	✓	✓	✓
Dependent Decisions				
Unique	✓			✓
Identical	✓	✓		✓
Distinct Symbols	✓			✓
Concurrent	✓		✓	✓

Although we did conduct experiments for all combinations of working memory model, dimension of task, and learning algorithm (where learning algorithm refers to whether or not eligibility traces were used), in many cases we omitted presenting results for particular combinations. For example, if agents learning to use a memory model without eligibility traces provided results from which we could draw significant conclusions, then we omitted results for agents learning with eligibility traces. Likewise, if results for agents learning with eligibility traces were sufficient in order to understand how agents were able to learn to use a memory model for a certain parameterization of task, then we omitted results for agents that learned without eligibility traces.

We omit the results gathered in TMaze tasks with second-order knowledge because the parameterizations in those tasks were closely related to the number of distinct symbols and dependent decisions.

We present a summary of our empirical results in Table 11.2. This summary categorizes the trends that we observed across a parameterization of task dimension in Chapter IV. In the table, Bit(0) and Gated(0) indicate the respective memory model was combined with pure temporal difference Q-learning; Bit(λ) and Gated(λ) indicate that the respective memory model was combined with Q-learning and eligibility traces; and BitC(0) and GatedC(0) that internal actions were prohibited in the choice location and that Q-learning with pure temporal difference updates was used.

Table 11.2: Summary of empirical results and how performance scales along task dimensions in the TMaze domain.

	Quadratic	**Exp.**	**Sub-optimal**
Temporal Delay			
Unique	Gated(0)		Bit(λ)
Identical			Gated(0), Gated(λ), GatedC(0) Bit(0), Bit(λ)
Num. Decisions			
Unique		Gated(0)	Bit(λ)
Identical			Gated(0), Gated(λ) Bit(λ)
Num. Distinct	Gated(0)		Bit(λ)
Concurrent	Gated(0)		Bit(λ)

The top row of the table are categories of how learning performances relate to task dimension. Quadratic and exponential refer to how time to convergence scales relative to the task parameterization. When agent performances did not converge to optimal across all task parameterizations, then they are listed in the "sub-optimal" column; this category includes any behaviors that do not all converge to the optimal level of performance. Having summarized trends observed across parameterized task dimensions for combinations of memory model and learning algorithm, we can discuss trends of interest.

In the base TMaze task, all conditions except bit memory without eligibility traces converges to optimal. In fact, bit memory without eligibility traces does not perform much better than a baseline comparison, which is why we used agents that learned to use bit memory in conjunction with eligibility traces throughout our experiments.

The dimensions of task in which agent performance scales most favorably are those in which the time to convergence scales quadratically with the task parameterization. This includes temporal delay with unique observations, number of distinct symbols, and concurrent salient symbols. For agents using gated memory in these three task dimensions, the parameterization of task was not increasing the partial observability or the degree to which states were aliased in the task, and thus the temporal difference algorithms could very quickly find optimal behaviors in the tasks.

After those three task dimensions, the exponential scaling of time to convergence is the next best performing category. Agents using gated memory demonstrated exponential time to convergence for number of dependent decisions with unique observations. As the number of dependent decisions with unique observations increases, learning performance won't scale well, but agents converge to optimal behaviors given sufficient experience in the task.

When observations are aliased, though, then learning performance doesn't converge to optimal even when learning to use gated memory with eligibility traces. As mentioned in Section 7.2.2, identical observations in the decision points prevents an agent from appropriately assigning credit and blame to selected actions that depend on knowledge of salient symbols. The state aliasing has the effect of removing any history from the agent's internal state, so it cannot update actions that lead to aliased states appropriately. When eligibility traces aren't used, an agent cannot learn at all; even when they are, asymptotic performance decreases as the number of decisions increase.

We know which task and memory combinations have solution paths that must traverse aliased states from the results of our state diagram analysis. More broadly, the results of our state diagram analysis can be unified with the results shown in Table 11.2. Our state diagram analysis found that for all combinations of task with bit memory, the optimal path in the state diagram traversed aliased states; and for these combinations, the agents exhibited sub-optimal behaviors.

In the case of gated memory, state diagram analysis found that the optimal behavior avoided traversing aliased states in four task parameterizations: TMazes with extended temporal delay and unique observations, TMazes extended with additional dependent decisions and unique observations, TMazes extended with additional distinct observations, and TMazes extended with concurrent knowledge. Agent behaviors for these conditions all converged to optimal, either in experience that scaled quadratically or exponentially to the parameterization.

State diagram analysis similarly found that optimal behavior did traverse aliased states in the remaining task conditions of temporal delay with identical observations and number of dependent decisions with identical observations. Agent behaviors for these conditions was sub-optimal.

Given these results, we conclude that if the state diagram analysis for a given task and memory condition demonstrates that the optimal behavior avoids traversing aliased states, then an agent will be able to learn the optimal behavior. However, our results from the toy POMDP in Section 7.2.2.1 demonstrate that the inverse does not always hold: if a state diagram analysis does traverse aliased states, an agent may still consistently converge to the optimal behavior. We note that the exception was found only in a specially constructed POMDP, and not for the state diagram that resulted from any combination of memory model and task dimension.

Across all parameterizations that we explored, agents learning to use bit memory with eligibility traces consistently demonstrate behaviors in which asymptotic performances decrease as tasks increase in complexity. For tasks that have low values along the dimensions of how using memory relates to task, bit memory used in conjunction with eligibility traces may provide sufficient power; however, gated working memory would converge to optimal in many of the same tasks and require either as much or less experience to do so.

The relative performances of gated memory relate to the trends in performances observed in agents learning to use bit memory in the same set of tasks. Although asymptotic performance decreases along with increasing parameterizations of task for agents learning to use bit memory, the relative decrease in performance is less for temporal delay with unique observations, number of distinct symbols, and concurrent observations than it is for temporal delay with identical observations and number of decisions with either unique or identical observations.

For both types of memory, performance is poor for additional decisions in the identical observation case. When designing environments, conditioning decision points on salient knowledge and creating states aliased such that an agent cannot recover knowledge of history from the observations made after those dependent decisions should be avoided in the interest of agent performance.

11.2 Contrasting Gated and Bit Memory Models

While the goal of this thesis is to improve our broad understanding of how a reinforcement learning agent can learn to use memory in a task, it is also the first study of which we are aware that has performed an empirical comparison of the performances that are afforded to agents learning to use gated and bit memory models. The differences in learning performance are significant and consistent: agents using gated memory are more capable in partially observable reinforcement learning tasks and learn stable behaviors more quickly.

Consider the performances shown in Figure 5.1. When proposing this work, one initial concern was that the performances of gated and bit memory would be too similar, and that there was no reason to explore both memory models; instead we considered exploring only a single model of working memory and extrapolating the results to the entire class of working memory models. Thankfully we explored both, but it serves as a useful anecdote to remind us how often our machine learning instincts can be wrong.

Table 11.2 also provides a useful visualization of the broad differences in performances afforded by the two memory models. Note that for all parameterizations of task (not including the degenerate base TMaze case), agents using bit memory are always categorized as decreasing or baseline. Gated memory, on the other hand, is characterized as decreasing only twice, and baseline once; in all other cases agent performances uniformly converge to the optimal behavior.

Although the bit memory models that we explored in this thesis were sufficient to capture all relevant features of history in order to perform the investigated tasks, gated memory had additional architectural characteristics that helped constrain an agent's search through possible behaviors. These architectural constraints on the space of behaviors improved both time to convergence and agents' asymptotic performances.

As discussed in Section V, the architectural constraints that give rise to improved performances are derived from the perceptually grounded knowledge that is stored in gated memory. Storing grounded knowledge allows agents to only experience a subset of possible internal states for a particular task and memory combination, whereas storing abstract knowledge allows agents to experience all possible internal states. Furthermore the subset of states that an agent can experience in an episode either contain the salient symbol and thus are directly associated with the optimal behavior, or contain no relevant information about the salient symbol and are irrelevant to optimal behavior.

Storing grounded knowledge in memory, then, avoids one difficult problem that can arise when learning to use memory: learning an association between knowledge in memory and salient symbols observed in the environment. Gated memory has an architectural constraint that affords an agent an a priori relationship between salient symbols and memory contents by simply storing those symbols directly.

While gated memory avoids this particular problem that arises when learning to use memory, agents endowed with gated memory must still overcome the aforementioned chicken and egg problem of learning to use memory by simultaneously learning behavior over memory and in the environment. The difficulty of this problem is lessened considerably when using gated memory and by storing grounded knowledge. When the state diagrams for gated memory agents have optimal policies that traverse only completely observable states, agents always converge to optimal behaviors for all parameterizations of task that were explored. The only tasks where an agent using gated memory does not converge to optimal were tasks with temporal delay and identical observations, and number of dependent decisions and identical observations. In both cases, the identical observations lead to state diagrams in which all behaviors, including the optimal behaviors, traverse aliased states. Thus, while storing grounded knowledge in memory can overcome certain types of partial observability, it is not a universal solution.

Agents learning to use bit memory, on the other hand, must learn to associate knowledge in memory with salient symbols observed in the environment, all while simultaneously learning to perform in the task and learning to control memory. As the state diagrams illustrated in the previous chapter make clear, learning to associate knowledge in memory with salient symbols causes significant aliasing in the state space for a task and memory combination. This aliasing is a result of the lack of architectural constraints on how knowledge in memory can be stored, and the primary effect is that it allows agents using bit memory to interfere with previously learned behaviors.

We discussed these interference effects in Section 5.1.2. As agents learn to use bit memory, they begin to associate one setting of memory contents with a stable behavior in the environment: for example, when an agent observes A, setting memory contents to 1 and then selecting the correct corresponding terminal action. The problem is that as the state-action values for that behavior begin to back up positive reinforcement, an agent then desires to select those state-actions in all situations, even when it has not observed salient symbol A but instead some other symbol. An agent sets memory so that it may select the highly rewarding actions, but then receives negative reward, driving down the state-action values that had previously been learned (and were an optimal behavior for a portion of the state-action space).

We explored architectural modifications to bit memory that could help to avoid these issues. Of those modifications, preventing the selection of internal actions after having observed the salient symbol was the most effective. If one is implementing an agent that learns to use abstract knowledge in memory, then learning performance would be improved if agents were prevented from modifying memory contents after salient symbols had been observed and before all decisions that depended on those symbols had been made. Depending on the problem domain, this may range from trivial to very complex.

Overall, gated memory (or other memories that store concrete, perceptually grounded knowledge) has clear advantages to bit memory and other memories storing abstract knowledge. The performance advantages of gated memory over bit memory were observed in all tasks and across all experiments.

Importantly, the advantages of gated memory over bit memory are not because of the size of the possible action space that is possible in any state diagram. Consider the case of gated and bit memories in the base TMaze task, where gated memory afforded very clear benefits to a learning agent. In the state diagrams for the respective conditions (Figures 5.2 and 5.3), it is clear that in both cases there are 24 possible transitions (including the initial ϵ-transitions to the starting states). When not counting initial ϵ-transitions, then there are fewer possible transitions for an agent learning to use bit memory than there are for an agent learning to use gated memory. Therefore, the differences between the two memory models cannot simply be explained by the differences in the number of possible actions that an agent must explore and learn values over. Rather, the differences are in how the resulting transitions of the state diagram are structured (whether they create loops in the state space) and whether the transitions for the optimal policy traverse aliased states.

11.3 Evaluating Online Learning Performance

Whenever possible, we evaluated the mean learning performances of agents with respect to optimal. If all agent conditions converged to the optimal behavior, then we evaluated how time to convergence related to parameterized task dimensions. If agent performances did not all converge to optimal, then either we described qualitatively how quickly learning performance degraded relative to optimal as the task parameterization increased, or we conducted individual analyses to characterize how convergent behaviors changed with task parameterization.

Using the optimal behavior as a conceptual yardstick is useful because it is an objective measure; agents cannot perform better than optimally, and any deviation from optimal is undesirable. However, there are plenty of cases when optimal performance is *not* the right evaluation criteria. If a cognitive modeler were creating reinforcement learning agents that used memory, they would be concerned about whether the artificial agents exhibited behaviors similar to human subjects. If a researcher were developing artificial long- lived agents, they might be unconcerned with small deviations from optimal performance so long as the agents achieved certain milestone goals, such as bearing offspring, satisfying internal drives, and persisting in an artificial environment for at least a certain period of time.

Even as we use optimal behaviors as a measuring stick for evaluating agent performance, then, it is important to remember that in many cases near-optimal performance is perfectly sufficient for an agent designer's intended purpose.

11.4 Generality of Results

The focus of this thesis was on how online agents learn to use memory via direct interaction with memory and environment. This setting allows us to explore agents that have an artificial lifetime in which they learn, without using any models to perform simulations of environment dynamics (as in model-based reinforcement learning algorithms) or to sample the space of possible behaviors by making use of domain knowledge provided to the agent outside of its direct experience (as in policy search algorithms). This restricts our setting to the foundational online temporal difference learning algorithms.

After finding that agents using Q-learning converged to optimal behaviors more quickly than those using Sarsa in many combinations of parameterized TMaze task and memory model, we selected Q-learning as the algorithm that we use throughout the work presented in this dissertation. This affords the practical benefits of increased granularity when collecting data points across parameterized TMaze tasks, as well as incurring lower computational cost when running experiments.

Our results generalize across the class of online temporal difference learning algorithms: Q-learning, Sarsa, and model-free actor-critic temporal difference learning algorithms (Sutton & Barto, 1998). Common to all of these algorithms is the way that an agent learns to interact with its environment and the absence of any offline simulation of environmental dynamics. The issues that arise when learning to use memory, such as the chicken and egg problem, arise for all online model-free algorithms.

While the underlying dynamics that exist between memory dimensions and task characteristics when using online temporal difference algorithms also exist when in the model-based setting, it remains to be seen whether the offline setting introduces biases to how knowledge in memory is associated with a behavioral policy that change the way learning performances scale. Exploring learning to use memory in model-based approaches is one potential direction of future work.

We make several assumptions to simplify the space of possible task dimensions that we consider throughout this work. Our framework only supports Q-learning agents with tabular value-function representations, and we do not explore continuous action or continuous observation spaces, thus obviating any need for function approximation. As function approximation is used to cope with the curse of dimensionality via generalization and is not used as a panacea to the dynamics that can arise in a task (Sutton & Barto, 1998), our findings should extend to task and memory combinations in which function approximation is used. However, tasks with continuous action or observation spaces lack the discrete representation of task parameterizations that we used throughout our work. Therefore, it could be more difficult to determine the numerical parameters along which the amount of experience required for convergence to optimal behaviors would scale.

11.5 Eligibility Traces

Throughout our empirical work, we compared agent learning performance across parameterized tasks with and without eligibility traces. Having presented the results of agents learning both with and without the benefit of eligibility traces, we now discuss why we explore pure temporal difference learning when agents using eligibility traces uniformly perform better.

As we state in Chapter I, the goal of this work is not to find the best algorithms for learning to use memory or create a framework that learns to use memory with higher learning performance than any other framework. Rather, the goal of this thesis is to improve our understanding of the relationships between memory and task characteristic while simultaneously learning to use memory. When an experimental condition was sufficient in order to explore the relationship between a memory model and task dimension, then we didn't always present results from the other condition (although we did run all experiments for both conditions).

In the case of bit memory, agent performance in the base TMaze case was sufficiently poor (Figure 5.1) that as tasks were extended along different dimensions, the only conditions in which agents outperformed the baseline lesion performance were when agents also used eligibility traces. Rather than explore agents using pure temporal difference learning, we studied learning with eligibility traces. Our reasoning was that these results are representative of how learning performance would scale without eligibility traces, and further, that anyone using a framework in which a temporal difference learning agent is endowed with bit memory would use eligibility traces in order to achieve acceptable levels of performances.

While agents using eligibility traces perform better than those that don't, eligibility traces do incur additional computational overhead. While theoretically pure implementations require additional memory storage linear with the number of state and action combinations of a problem, practical implementations maintain only a finite number of eligible state-action pairs and do not update state-action pairs after their eligibility decays below a threshold. In this case the additional space complexity is constant and scales only with the eligibility trace decay rate and the tasks's discount rate, while time complexity scales linearly beyond pure temporal difference learning algorithms (Sutton & Barto, 1998, p. 189).

A researcher might use pure temporal difference methods instead of algorithms with eligibility traces in order to model human and primate learning. While beyond the general scope of the thesis, there is strong neurological evidence that primate brains perform temporal difference updates (Niv, 2009; Schultz *et al.* , 1997), but the evidence for eligibility traces in the brain is weaker (Pan *et al.* , 2005). In particular, while there are widely plausible neurological mechanisms by which temporal difference updates could be computed in the brain, eligibility traces would require that neural circuitry be continuously or repeatedly activated on the order of seconds (Pan *et al.* , 2005). Thus, while eligibility traces can significantly improve learning performance in artificial agents, the results for pure temporal difference learning methods may also be relevant.

In every combination of memory and task in which agents learned to use memory with eligibility traces, we performed a parameter sweep to determine the optimal setting of the eligibility trace decay rate. Across all task and memory combinations we found that the optimal setting was $\lambda = 0.9$, and that behaviors were robust within a narrow range around that value.

11.6 On Numerical Parameters

Whether or not to use eligibility traces or not is one of many design decisions that an agent designer must make. Once the decision to use eligibility traces has been made, though, one must set the eligibility trace decay rate; the selection of numerical parameters is different from that of architectural and learning algorithm design decisions in that numerical parameters are typically selected in order to optimize performance.

In an ideal world, reinforcement learning algorithms would be parameter-free and frameworks would select and tune parameter settings online. While this is an active research direction, state of the art online algorithms have parameters that must be set by an agent designer.

Not shown in this thesis are the parameter sweeps that we conducted. Anecdotally we report that in all cases we found parameters that maximized performance to be relatively easy to find, and that the gradient of the parameter search space appeared to indicate that the local minima we selected were also global minima, although our discretized search space is no guarantee of that.

We also report that, although we used relatively consistent settings of most parameters across our empirical work, learning performances were robust across many of the parameters. The parameters for which this was not true included the initial

Q-values in agent value functions and the temperature for our softmax greedy action selection. If Q-values were not optimistically initialized, agent performance suffered dramatically; similarly, if the temperature for our softmax action selection procedure encouraged too much exploration, the time to convergence was significantly longer than in the results we presented.

Throughout all of our experiments presented in this thesis, we use a constant learning rate. In exploratory work, we examined the effects of a decaying learning rate in an agent learning to use bit memory in the base TMaze task, both with and without architectural modification. As expected, once an agent had converged to a stable behavior, decaying the learning rate had no discernible effect. Similarly, decaying the learning rate before an agent's behavior had converged to a stable equilibrium also had no discernible effect. As our initial explorations with decaying learning rates had no functional effects, we opted to explore agents using only fixed learning rates throughout our work in order to constrain the combinatorial effects of exploring multiple parameters simultaneously.

11.7 Implications for Cognitive Architectures

Cognitive architectures are attempts at modeling the computational bounds of human cognition in the same way that information processing is constrained in the human brain. Soar is a cognitive architecture that includes working memory, procedural knowledge, episodic memory, semantic memory and reinforcement learning as architectural modules (Laird, 2008). CLARION (Sun, 2006), like Soar, includes working memory, procedural knowledge, episodic memory, semantic memory and reinforcement learning modules amongst its architectural components (Sun *et al.* , 2011). ACT-R, another cognitive architecture, includes procedural, declarative, imaginal and goal modules, as well as a utility-learning mechanism that propagates reinforcement (Anderson, 2007).

While cognitive architectures include separate memory mechanisms as well as reinforcement learning functionality, the dynamics that arise when reinforcement learning is used to learn to use memory mechanisms have not been studied beyond our investigations of agents that learned to use Soar's episodic memory (Gorski & Laird, 2011).

The results from this work suggest that limiting the actions over a memory model that interact with reinforcement learning can result in significant improvements to performance, and thus the functionality of a learning agent. We demonstrated that by prohibiting internal actions at certain times during a task an agent's learning performance could be significantly improved; specifically between when a salient symbol was observed and when an environmental action was conditioned upon the identity of that salient symbol. By preventing an agent from selecting internal actions and thus interfering with any potential knowledge of that salient symbol, agents learned significantly better in the task.

This suggests that rather than make all actions over memory interact with an agent's reinforcement learning mechanism, there may be functional advantages to limiting which actions over memory can be learned. In particular, if salient symbols are automatically stored to long-term memory then agents could learn to use deliberate retrieval actions that place salient knowledge in immediate working memory.

This is very similar to how Soar's episodic memory mechanism is currently implemented (Derbinsky & Laird, 2009; Nuxoll & Laird, 2007). Storage to long-term memory is automatic and controlled by the underlying cognitive architecture, but retrievals are modulated by procedural knowledge that interacts with the reinforcement learning mechanism.

In previous work, we developed agents that learned to use Soar's episodic memory and demonstrated that for some tasks agents could successfully learn the optimal behavior, but for others agents learned a mix of optimal and suboptimal behaviors

(Gorski & Laird, 2011). Although architectural storage may significantly improve agent performance when learning to use memory, it is not be a panacea and additional work would be necessary to identify how learning performance when using long-term memory models scales with dimensions of task.

11.8 The Future of Learning To Use Memory

The historical origins of reinforcement learning agents endowed with bit memory mechanisms was an attempt to create online temporal difference agents that could learn to perform capably in partially observable domains. Initial work demonstrated that Sarsa(λ) could perform well in POMDPs with near-optimal memoryless policies, where a memoryless policy is a mapping of immediate observation to action (Loch & Singh, 1998). Other early work in tackling the problem of partial observability took the form of online temporal difference methods that learned to use bit memory, as discussed in Chapter II.

While online methods, both with and without memory, did demonstrate success in classes of problems, attention in the literature quickly focused on memory-based approaches, but not memory in the sense of our memory models. Instead, offline batch learning approaches learn a probabilistic belief state that is broadly applicable to the entire class of POMDPs.

It has been only recently that attention has been given to using reinforcement learning in conjunction with memory models other than bit memory. We offer three examples of fields that have recently seen interest in using reinforcement learning in conjunction with gated, episodic and semantic memories. First, as the computational models of memory and learning used by computational neuroscientists have matured, there has been more attention given to learning to use memory (O'Reilly & Frank, 2006; Todd *et al.* , 2008; Zilli & Hasselmo, 2008c). Second, as cognitive architectures have been extended with reinforcement learning, increased attention has focused on using multiple learning mechanisms in conjunction with one another (Laird, 2008; Gorski & Laird, 2011). Third, embodied robots have increasingly been using frameworks that are moving closer to full-fledged cognitive architectures, and as computational power of robots increases reinforcement learning becomes a more attractive thing to do on robots (Dodd & Gutierrez, 2005).

The initial focus on bit memory over gated memory is understandable: reinforcement learning is about abstract knowledge. Furthermore, as we observed in Gorski & Laird (2011), many partially observable tasks require only a single bit of information in order to completely capture all aspects of history and make the task fully observable to a reinforcement learning agent; this holds true for tasks used throughout the literature that explored learning to use memory. Given that bit memory was sufficient to represent an optimal policy, exploring alternative memory mechanisms with architectural constraints on behavior and additional representation capacity is an unintuitive direction. Thus only as statistical reinforcement learning, cognitive science and computational neuroscience communities have begun to mix has attention shifted to the capabilities that a perceptually grounded memory (such as gated memory) can afford an agent.

Given this recent attention to reinforcement learning with models other than bit memory, demonstrating that learning to use gated memory is significantly more tractable than bit memory, and the understanding of how performance scales, may encourage more attention to be given to the online partially observable tasks, with agents that use cognitively inspired memory models in order to achieve levels of performance that were previously unobtainable.

11.9 Contributions

To the best of our knowledge, this thesis is the first comprehensive study of how the online performance of learning to use working memory models is impacted as tasks become more complex along specific dimensions. We identify dimensions of task in which learning to use working memory scales favorably and unfavorably.

We also believe this to be the first study that has demonstrated that storing and encoding perceptually grounded symbols (as in gated memory) affords significantly better task performance than does storing abstract knowledge (as in bit memory) across a wide range of relevant dimensions of task. In this thesis we demonstrated that: agents learning to use gated memory converge to optimal behaviors significantly more quickly than agents learning to use bit memory; agents learning to use gated memory converge to optimal behaviors in many situations when agents learning to use bit memory do not; and agents storing perceptually grounded knowledge to memory simplifies the agent's search through the space of possible behaviors.

This thesis demonstrates that limiting the availability of certain internal actions can result in learning task performance. While agents can effectively learn optimal behaviors to control their memory and task actions, for certain memory and task combinations agents can not learn optimal behaviors unless certain actions that control memory are fixed in the agent's architecture. When learning to use both gated and bit memory, agent performance was improved in two tasks when agents were prohibited from selecting internal actions in states that would interfere with potential knowledge of salient symbols. When these potentially interfering actions were inhibited, agents demonstrated learning optimal behaviors when they otherwise learned only sub-optimal behaviors. This suggests that not all control over the storage and retrieval of information to and from memory should be in the form of internal actions that can be modulated by reinforcement learning, but rather that certain internal actions can be learned more efficiently if the agent's underlying architecture is responsible for certain aspects of memory control. In this work we demonstrated that architectural maintenance of knowledge can be beneficial, which is most similar to architectural storage actions, suggesting that retrieval actions could still interact with reinforcement learning mechanisms and result in desirable task performance.

We developed a methodology to analyze the online behavior that reinforcement learning agents exhibit that involves three steps: first, comparing optimal-normalized mean learning curves; second, describing the multi-modal aggregate behaviors of agents in each condition by categorizing behaviors that individual agents exhibit; third, analyzing state diagrams of the combined memory and task state space to characterize problems that prevent agents from learning optimal task behaviors.

In the case of our state diagram analysis specifically, we demonstrated that the entire state diagram must be analyzed and not just the sequence of states that the optimal behavior traces. In section 10.2, we created artificial POMDPs and demonstrated that modifying tasks by adding a terminal action that results sub-optimal reward would prevent agents from learning optimal behaviors. Analyzing the sequence of states along the path of optimal behavior is insufficient when those states are not perceptually unambiguous.

11.10 Future Work

Additional work would broaden the focus of learning to use memory to general memory models and tasks, and could extend our investigation to theoretical results.

This thesis examined two short-term working memory models: gated and bit memory. While prior work investigated learning to use a long-term episodic memory mechanism (Gorski & Laird, 2011), it did so for only a handful of tasks and in tasks that were not parameterized along specific dimensions. A principled empirical study of learning to use long-term memories would identify which memory actions could be learned online through direct experience, as well as those in which online learning is insufficient to converge to optimal behaviors.

Our investigation explored tasks parameterized across dimensions that relate specifically to how knowledge must be maintained in memory in order to perform the task.

In order to attempt to control for dependent variables, we considered tasks with specific reward functions and task dynamics (such as actions with deterministic effects). An exploration of learning to use memory across general POMDPs and the effects of modifying dimensions of task that do not relate specifically to learning to use memory would broaden our practical understanding of learning to use memory.

This work was an empirical investigation, and could be extended with additional theoretical results. Throughout our experimental results we identified relationships between episodes until convergence to optimal behavior and task parameters. While we identified closed form equations, we did not offer theoretical proofs of these relationships which is a direction for future work.

More generally, formally characterizing the space of POMDP tasks that can be solved by agents that learn to use memory is a natural goal for research investigating learning to use memory. Zilli & Hasselmo (2008b) demonstrated that a given POMDP and memory model (either gated memory or abstract episodic memory) it could be determined whether the memory mechanism was sufficient to support the optimal task behavior. However, we have demonstrated throughout this work and in related work (Gorski & Laird, 2011) that having sufficient representational capacity in memory is no guarantee that an agent can learn to use it optimally, or learn to perform in the task optimally.

One final question is whether learning to use memory scales beyond the scope of simple, extended TMaze tasks. When using gated memory, the time to learn optimal policies scaled either quadratically, exponentially, or did not always converge to optimal; clearly, if we were to use the same agent framework to tackle the every day domains of humans, agents would be unable to learn to use memory.

One potential answer would be generalization. If agents were capable of realizing an effective functional generalization over their Q-value representations, then learning performance might scale. We say a functional generalization because the specifics of how this generalization would be realized are inconsequential to the effects that it would have: whether the generalization is over hierarchical action representations, abstractions of objects in the task, generalizing over value functions, or any potential generalization. All of these potential generalization strategies (and many more) have been investigated in the context of reinforcement learning, and yet partial observability remains, in many ways, as a block towards tractably extending reinforcement learning to problems on the scale of those that humans face. In this work we explore the dynamics that arise when learning to use memory, but learning to use memory is no panacea to the overarching problem of partial observability.

More broadly, these results demonstrate that agents may learn to use working memory along certain dimensions, but as tasks scale in complexity along others agents are unable to effectively learn to use memory, suggesting that knowledge about when to select memory actions might be required from sources other than learning via direct experience in the domain as in reinforcement learning.

BIBLIOGRAPHY

BIBLIOGRAPHY

Anderson, John R. 2007. *How Can the Human Mind Occur in the Physical Universe?* New York: Oxford University Press.

Arai, S., & Sycara, K. 2001. Credit Assignment Method for Learning Effective Stochastic Policies in Uncertain Domains. *In: Proceedings of the genetic and evolutionary computation conference.*

Baddeley, A. D. 1986. *Working Memory.* Oxford: Clarendon Press.

Baird, Leemon, & Moore, Andrew. 1999. Gradient Descent for General Reinforcement Learning. *Pages 968–974 of: Advances in neural information processing systems 11.* MIT Press.

Derbinsky, Nate, & Laird, John E. 2009. Efficiently Implementing Episodic Memory. *In: Proceedings of the 8th international conference on case-based reasoning.* Seattle, WA: Springer.

Dodd, W, & Gutierrez, R. 2005. The Role of Episodic Memory and Emotion in a Cognitive Robot. *Pages 692–697 of: Proceedings of the 14th annual ieee international workshop on robot and human interactive communication (ro_man).*

Gluck, M. A., Mercado, E., & Myers, C. E. 2007. *Learning and Memory: From Brain to Behavior.* New York: Worth Publishers.

Gorski, Nicholas A., & Laird, John E. 2009. Learning to Use Episodic Memory. *In: Proceedings of the 9th international conference on cognitive modeling.*

Gorski, Nicholas A., & Laird, John E. 2011. Learning to use episodic memory. *Cognitive systems research*, **12**(2), 144–153.

Laird, John E. 2008. Extending the Soar Cognitive Architecture. *In: Proceedings of the first conference on artificial general intelligence (agi-08).*

Lanzi, P. L. 2000. Adaptive Agents with Reinforcement Learning and Internal Memory. *In: Sixth international conference on the simulation of adaptive behavior.*

Lanzi, P. L., & Wilson, S. W. 2000. Toward Optimal Classifier System Performance in Non-Markov Environments. *In: Evolutionary computation.*

Littman, Michael L. 1994. Memoryless policies : theoretical limitations and practical results. *Pages 238–245 of:* Cliff, D, Husbands, P, Meyer, Jean-Arcady, & Wilson, Stewart W (eds), *From animals to animats 3 proceedings of the third international conference on simulation of adaptive behavior.* MIT Press.

Loch, John, & Singh, Satinder. 1998. Using Eligibility Traces to Find the Best Memoryless Policy in Partially Observable Markov Decision Processes. *Pages 323–331 of: Proceedings of the fifteenth international conference on machine learning.* Morgan Kaufmann Publishers Inc.

Malone, J. C. 1991. *Theories of Learning: A Historical Approach.* Belmont, CA: Wadsworth Publishing Company.

Meuleau, N., Peshkin, L., Kim, K. E., & Kaelbling, L. P. 1999. Learning Finite-State Controllers for Partially Observable Environments. *In: Uncertainty in artificial intelligence.*

Niv, Yael. 2009. Reinforcement learning in the brain. *Journal of mathematical psychology*, **53**(3), 139–154.

Nuxoll, Andrew M, & Laird, John E. 2007. Extending Cognitive Architecture with Episodic Memory. *In: Proceedings of the 22nd national conference on artificial intelligence (aaai).*

O'Reilly, R. C., & Frank, M. C. 2006. Making working memory work: A computational model of learning in the prefrontal cortex and basal ganglia. *Neural computation*, **18**, 283–328.

Pan, Wei-Xing, Schmidt, Robert, Wickens, Jeffery R, & Hyland, Brian I. 2005. Dopamine cells respond to predicted events during classical conditioning: evidence for eligibility traces in the reward-learning network. *The journal of neuroscience*, **25**(26), 6235–6542.

Peng, J, & Williams, R. J. 1996. Incremental multi-step Q-learning. *Machine learning*, **22**(1/2/3).

Peshkin, Leonid, Meuleau, Nicolas, & Kaelbling, Leslie Pack. 1999. Learning Policies with External Memory. *Pages 307–314 of: Proceedings of the sixteenth international conference on machine learning.* San Francisco, CA: Morgan Kaufmann Publishers Inc.

Rummery, G. A., & Niranjan, M. 1994. *On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166.* Tech. rept. Engineering Department, Cambridge University.

Schultz, W., Dayan, P., & Montague, P. Read. 1997. A Neural Substrate of Prediction and Reward. *Science*, **275**(5306), 1593–1599.

Sun, R. 2006. *The CLARION cognitive architecture: Extending cognitive modeling to social simulation.* New York: Cambridge University Press.

Sun, Ron, Helie, Sebastien, & Wilson, Nick. 2011. *The CLARION Cognitive Architecture: A Tutorial (Part 1: Introduction).*

Sutton, Richard S., & Barto, Andrew G. 1998. *Reinforcement Learning: An Introduction.* MIT Press.

Todd, Michael T, Niv, Yael, & Cohen, Jonathan D. 2008. Learning to use Working Memory in Partially Observable Environments through Dopaminergic Reinforcement. *In: Proceedings of nips 2008.*

Tolman, E. C., & Honzik, C. H. 1930. Degrees of Hunger, Reward, and Nonreward, and Maze Learning in Rats. *University of california publications in psychology*, **4**, 215–232.

Watkins, C. J. C. H. 1989. *Learning From Delayed Reward.* Ph.D. thesis, Cambridge University.

Whiteson, Shimon, & Littman, Michael L. 2011. Introduction to the special issue on empirical evaluations in reinforcement learning. *Machine learning*, **84**(June), 1–6.

Zilli, E. A., & Hasselmo, M. E. 2008a. The Influence of Markov Decision Process Structure on the Possible Strategic Use of Working Memory and Episodic Memory. *Plos one*, **3**, e2756.

Zilli, Eric a, & Hasselmo, Michael E. 2008b. Analyses of Markov decision process structure regarding the possible strategic use of interacting memory systems. *Frontiers in computational neuroscience*, **2**(December), 6.

Zilli, Eric A, & Hasselmo, Michael E. 2008c. Modeling the Role of Working Memory and Episodic Memory in Behavioral Tasks. *Hippocampus*, **2**(18), 193–209.