EFFECTIVE REINFORCEMENT LEARNING THROUGH INTRINSIC MOTIVATION AND VISUAL EXTERNAL MEMORY IN PARTIALLY OBSERVABLE ENVIRONMENTS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

BURAK HAN DEMIRBILEK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2023

Approval of the thesis:

EFFECTIVE REINFORCEMENT LEARNING THROUGH INTRINSIC MOTIVATION AND VISUAL EXTERNAL MEMORY IN PARTIALLY OBSERVABLE ENVIRONMENTS

submitted by **BURAK HAN DEMIRBILEK** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _______________

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** _______________

Prof. Dr. Faruk Polat
Supervisor, **Computer Engineering, METU** _______________

Assist. Prof. Dr. Alper Demir
Co-supervisor, **Computer Engineering, IUE** _______________

Examining Committee Members:

Prof. Dr. Göktürk Üçoluk
Computer Engineering, METU _______________

Prof. Dr. Faruk Polat
Computer Engineering, METU _______________

Assoc. Prof. Dr. Mehmet Tan
Computer Engineering, TOBB ETU _______________

-
- _______________

-
- _______________

Date:08.09.2023

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname:	Burak Han Demirbilek
Signature :	

ABSTRACT

EFFECTIVE REINFORCEMENT LEARNING THROUGH INTRINSIC MOTIVATION AND VISUAL EXTERNAL MEMORY IN PARTIALLY OBSERVABLE ENVIRONMENTS

Demirbilek, Burak Han
M.S., Department of Computer Engineering
Supervisor: Prof. Dr. Faruk Polat
Co-Supervisor: Assist. Prof. Dr. Alper Demir

September 2023, 106 pages

Reinforcement learning in practical scenarios often includes partial observability that requires long-term remembering of visual observations to obtain optimal policies. Addressing this challenge, this study introduces agents augmented with visual external memories, enhancing agents decision-making capabilities by constructing a context derived from both current observations and memory data. Moreover, to ensure effective utilization of the external memory for the agent, intrinsic motivation is incorporated as a secondary reward system, promoting long-term beneficial behaviors of using memory. Key contributions from this study include a novel framework for integrating visual external memory in reinforcement learning agents, the development of intrinsic motivation functions to efficiently learn how to utilize external memory to improve overall learning, empirical evaluations and experiments in various environments, and detailed comparison and analysis against the state-of-the-art. The results highlight the potential and advantages of the proposed approaches and present numerous possibilities for future investigation within this particular field of study.

Keywords: Reinforcement Learning, Partial Observability, External Memory Management, Intrinsic Motivation

ÖZ

KISMİ GÖZLEMLENEBİLİR ORTAMLARDA İÇSEL MOTİVASYON VE GÖRSEL HARİCİ BELLEK İLE ETKİLİ PEKİŞTİRMELİ ÖĞRENME

Demirbilek, Burak Han
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Faruk Polat
Ortak Tez Yöneticisi: Dr. Öğr. Üyesi. Alper Demir

Eylül 2023 , 106 sayfa

Gerçek hayattaki pratik pekiştirmeli öğrenme problemlerinde, problemlerin kısmi gözlemlenebilir olduğu ve optimal çözümler için bazı gözlemlerin uzun süreli hatırlanması gereken durumlarla sıklıkla karşılaşılmaktadır. Bu zorluğa çözüm oluşturulması amacıyla, etmenlere içeriğini kendilerinin yönetebileceği harici görsel bellek mekanizmaları oluşturulup, karar verme kabiliyetleri genişletilmiştir. Geliştirilen bu görsel bellek yönetimi yaklaşımı sayesinde, mevcut gözlem ve harici bellek verileri kullanılarak kısmi gözlemlenebilir senaryolar için etmenlerin bağlamlar oluşturması sağlanabilmektedir. Ayrıca, etmenin bu harici belleği uzun vadede etkili bir şekilde kullanmasını teşvik edebilmek amacıyla, içsel motivasyon yaklaşımları bu problemler için ikincil bir ödül mekanizması olacak şekilde dahil edilmiştir. Bu çalışma sonucu elde edilen ana katkılar arasında, pekiştirmeli öğrenme etmenleri için görsel harici bellek yönetimi çerçevesinin oluşturulması, harici belleğin verimli biçimde kullanılmasına teşvik edilebilmesi amacıyla görsel içsel motivasyon fonksiyonlarının geliştirilmesi, farklı ortamlar üzerinde deneyler gerçekleştirilmesi ve literatürdeki en başarılı yaklaşımlarla karşılaştırılması yer almaktadır. Elde edilen sonuçlar, önerilen yaklaşımların potansiyelini ve avantajlarını vurgulamakta ve bu özel çalışma alanı içinde gelecekteki araştırma için çok sayıda olasılığı okuyucuya sunmaktadır.

Anahtar Kelimeler: Pekiştirmeli Öğrenme, Kısmi Gözlemlenebilirlik, Harici Bellek Yönetimi, İçsel Motivasyon

To my family.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Prof. Dr. Faruk Polat, for his patience, continuous support, and guidance throughout the course of this work. His insights and wisdom were invaluable, and I feel fortunate to have had the opportunity to work under his supervision. Prof. Polat created the perfect research environment for me, where I could both be guided by my professors and have opportunities for research and self-development.

I would also like to thank my co-supervisor, Dr. Alper Demir, for including me in his research project and partially funding my studies. I also want to express my gratitude for his endless support, patience, and close interest in my thesis studies. He always had time for me, and I am very glad to have had the opportunity to work with him.

Finally, I would like to thank the remaining members of our research group, Dr. Erkin Çilden, Dr. Hüseyin Aydın, and Saim Sünel, for their support and valuable contributions. I thank my family and friends for their endless patience and for always motivating me. Lastly, I also want to thank my close friend Batu Kaan Özen, whom I have known since my Bachelor's studies, for his unwavering support.

This work is partially supported by the Scientific and Technological Research Council of Turkey under Grant No. 120E427.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

In the context of Artificial Intelligence (AI), an agent is an entity that perceives its environment through sensors and acts upon that environment with actuators to achieve specific goals. More specifically, an agent defines a policy that maps from a given state or observation of the environment to an action. Conversely, the environment responds to the agent's actions, providing new observations and rewards, thereby encapsulating the dynamics and challenges the agent must navigate. Together, they form a feedback loop where the agent continually adjusts its behavior based on the consequences of its actions in the environment.

Reinforcement Learning (RL) is an area focused on optimal control and decision-making in Markov Decision Processes (MDPs). It involves formulating agents that interact with environments to learn policies that maximize a given optimization objective. While some RL methods operate on fully specified MDPs using the given transition dynamics and reward structures, others learn policies through direct interactions in scenarios where the MDP is incompletely known.

However, in real-world scenarios, the entire state of the environment is often not completely visible to the agent at all times, rendering these situations as Partially Observable Markov Decision Processes (POMDPs). In such circumstances, the agent is required to make decisions based solely on its current observation, which only offers a partial view of the entire state. In practice, there are many types of partial observability, and there is no perfect method to optimally solve all kinds within a single method; even many cannot be solved due to the complexity, which becomes computationally intractable in practice.

Moreover, this work focuses on a specific subset of POMDP problems that require long-term remembering of visual observations to obtain optimal policies. With this problem type, some form of context or memory can be used to augment this POMDP problem into a Hidden State MDP, which can be optimally solved in theory. Unfortunately, due to computational limitations and approximation errors, solving POMDP problems that require long-term remembering of visual observations is very challenging and easy to become intractable in practice.

In this study, agents are equipped with visual external memories, enabling memory context to be perceived alongside their environmental observations. The manipulation of this external memory is governed by the agents themselves, and this can beneficially influence their future behaviors.

While external memory provides a mechanism to retain historical information, there arises a need for a guiding force to encourage efficient memory utilization. This is where intrinsic motivation comes into play. Intrinsic motivation serves as an internal reward system, pushing the agent towards behaviors that are not just immediately rewarding but also beneficial in the long run, such as effective memory management.

1.1 Contributions

The aim of this thesis has been to bridge the gaps in the realm of partially observable reinforcement learning, particularly intrinsic motivations and the strategic utilization of visual external memories. This work has led to several important contributions:

- **Framework for visual external memory integration**: One of the main contributions of this work is the novel framework designed to integrate visual external memory with reinforcement learning agents seamlessly. This design ensures agents are better equipped to make decisions in partially observable environments by leveraging an external memory.
- **Intrinsic motivation functions for Visual Memory Management**: Recognizing the critical role of intrinsic motivation in decision-making processes, a unique set of intrinsic motivation functions were developed for visual tasks.

These functions, tailored for agents with visual memory management, ensure that memory utilization is optimized to benefit decision-making strategies.

- **Insights into the complexity in exploration**: With the introduction of visual external memory, agents expose the 'curse of dimensionality' due to the growing observation and action spaces. This study provides a deep dive into the results of extending the action and observation space, offering strategies to overcome this negative effect.
- **Open source codes of the algorithms and experiment results:** The entire work is open-sourced for future researchers using the same framework to utilize the results for their research. The comparative analysis is included to assess the performance of Visual Self Memory Management (VSMM) and IM approaches against the state-of-the-art. Source codes can be accessed at https://github.com/burakdmb/vsmm.

1.2 Outline

The subsequent chapters are organized as follows:

- **Chapter 2** provides a comprehensive review of the current state-of-the-art in RL, POMDPs, approaches for solving POMDPs such as Belief State and Hidden State POMDPs, Deep RL, Neural Network architectures, intrinsic motivation approaches, and lastly, related works.
- **Chapter 3** delves into the methodologies employed, providing details about the design and implementation of proposed intrinsically motivated agents with external visual memory.
- **Chapter 4** presents the experimental setup, including the environments selected and the metrics used for evaluation, discusses the experiment results, and compares with state-of-the-art methods.
- **Chapter 5** concludes the thesis, summarizing the findings and suggesting avenues for future research.

CHAPTER 2

BACKGROUND INFORMATION AND RELATED WORKS

2.1 Markov Decision Processes (MDPs)

Markov Decision Processes (MDPs) provide a mathematical framework for modeling discrete time stochastic control/decision-making processes. Foundations of MDPs date back to the late 1950s [7] and 1960s [8] with academic pioneers Prof. Richard Bellmann (mostly known with the Bellmann Equation and contributions in value iteration approaches) and Prof. Ronald A. Howard (mostly known for contributions to policy iteration approaches). The name comes from the Russian mathematician Andrey Markov as MDPs are an extension of the Markov Chains (and therefore based on Markov Property). The Markov Property is characterized by the principle that the probability of a subsequent event depends solely on its immediate predecessor and is independent of all preceding events. If an event exhibits this characteristic, it is said to be 'Markovian' or to possess the Markov Property. Based on this definition, a Markov Chain can be formalized as a stochastic process in which every event adheres to the Markov Property.

MDPs are modeled with a four tuple, $\langle S, A, T, R \rangle$ where:

- S is the finite set of states,
- A is the finite set of actions that the agent can take,
- $T : S \times A \times S \to [0, 1]$ is the transition function,
- $R : S \times A \times S \to \mathbb{R}$ is the reward function.

For demonstration, the state transition graph of a simple example MDP is given in

Figure 2.1.

Furthermore, in a more abstract and natural view, an MDP can be viewed as agents acting in an environment with defined states, actions, transition probabilities, and rewards (Illustration is shared in Figure 2.2). An agent can be a decision-making human, a robot, or a control algorithm. As can be seen, many tasks/problems in nature (decision-making of humans) or science can be modeled with MDPs, and with appropriate methods, MDPs can also be solved. In addition, MDPs are characterized by full observability, which means that the agent has complete knowledge of the state of the environment at every time step.

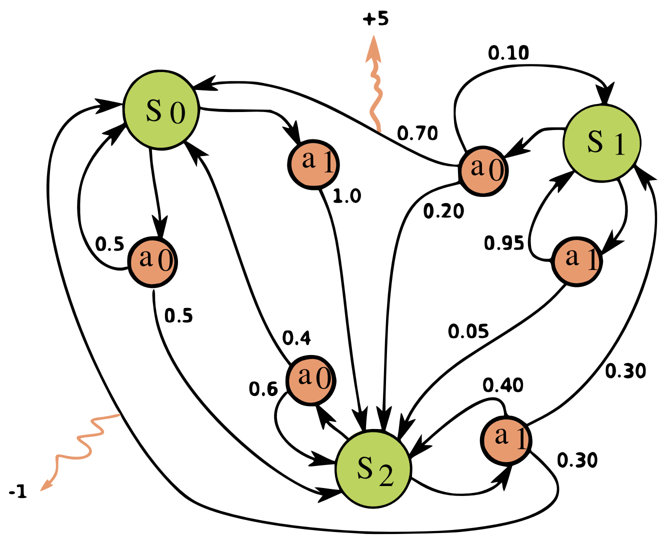


Figure 2.1: An example state transition graph for a Markov Decision Process with three states (green cells: s_0, s_1, s_2) and two actions (orange cells: a_0, a_1). Black arrows denote the transition probability from one state to another, and orange arrows denote the rewards obtained from transitioning from one state to another with the given action. Example taken from: [2], best viewed in color.

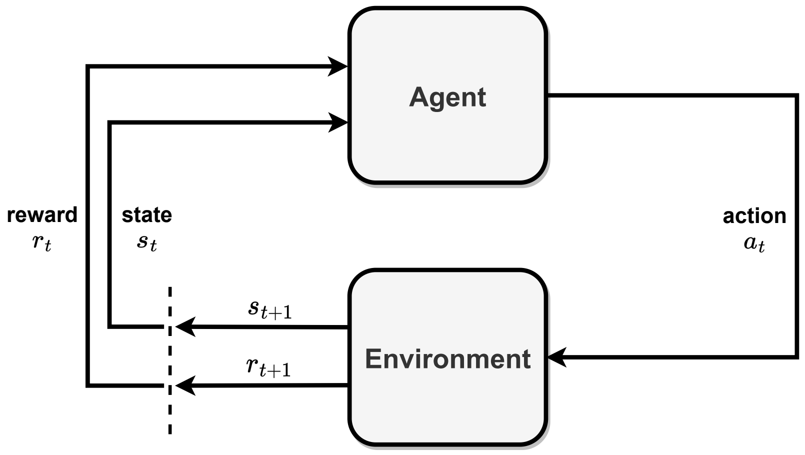


Figure 2.2: Illustration of a Markov Decision Process, abstract perspective with the concepts of agent and environments. Figure adapted from: [3].

In a finite MDP, all defined random variables, such as states, actions, and rewards, will have a finite number of elements as the agent interacts with the environment at each discrete time step, $t = 0, 1, 2, ...$ This results in a sequence of MDP variables such as:

$$\{s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, ...\}$$

If the maximum time is finite instead of ∞, this form of a finite sequence is named an Episode of the MDP. When T is the final time step of this episode, E is defined as:

$$E = \{s_0, a_0, r_1, s_1, a_1, ..., r_{T-1}, s_{T-1}, a_{T-1}, r_T\}$$

For arbitrary values of these random variables, $s' \in S$ and $r \in R$, the dynamics of the MDP can be defined as:

$$p(s', r|s, a) = Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\} \tag{2.1}$$

for all $s', s \in S, r \in R$ and $a \in A(s)$.

2.2 Reinforcement Learning

The problem of reinforcement learning is defined as the optimal control of incompletely-known Markov Decision Processes by using theoretical foundations of dynamical systems theory and computer science [3]. Reinforcement learning uses techniques such as dynamic programming to obtain solutions for unknown MDPs.

In reinforcement learning, the agents aim to obtain optimal rewards from the environment itself. At each time step, the agent tries to optimize its total rewards, and the optimality of rewards means the maximization of cumulative rewards in the long run. The agent behavior function, which designates the selection probability of each feasible action given the state and is referred to as the 'Policy', is also introduced. It is denoted as π. Through the policy function π, the agent's behavior within the current MDP is determined, resulting in the acquisition of rewards corresponding to their respective actions. The agent's policy can be characterized as the probability function $\pi(a|s), a \in A_t, s \in S_t$.

In control theory, the optimization criteria of rewards (or in this case, costs) mainly refers to minimization, while in computer science and artificial intelligence literature, it refers to maximization. The notation employed in this study adheres to the maximization of cumulative rewards notation for an agent's objective. The cumulative sum of rewards can be denoted as the expected total return, G_t, and can be defined as follows:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T \tag{2.2}$$

Where T represents the final time step. Since not all MDPs are finite in step length, it is possible to extend the concept of T to infinity by incorporating a discount rate denoted as γ, satisfying the condition $0 \leq \gamma \leq 1$.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.3}$$

If $\gamma < 1$, the infinite sum value will be finite. As γ approaches zero, the effects of the future returns will be weaker, and as γ approaches one, the effect of the future rewards is more substantial.

Expected return G_t can also be written as a recursive form:

$$G_t = R_{t+1} + \gamma G_{t+1} \tag{2.4}$$

Additional expressions can be defined with the definition of expected return. Firstly, a function can be defined to estimate the expected return of a specific state when an agent follows a policy π. This is named as "State Value Function" and denoted as $v_\pi(s)$. It can be defined as:

$$V_\pi(s) := \mathbb{E}\left[G_t | S_t = s\right] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right] \tag{2.5}$$

Equation 2.5 can also be written in a recursive form, given in Equation 2.6. This is the Bellmann Equation form for V_π. This constructs the relationship between the value functions of the current and successor states, and this recurrent relationship is the crucial element for algorithms to compute, approximate, or learn V_π.

$$\begin{aligned}
V_\pi(s) &= \mathbb{E}\left[G_t | S_t = s\right] && \text{(2.6a)}\\
&= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} | S_t = s\right] && \text{(2.6b)}\\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a) \Big[r + \gamma \mathbb{E}_\pi \left[G_{t+1} | S_{t+1} = s'\right] \Big] && \text{(2.6c)}\\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a) \Big[r + \gamma V_\pi(s') \Big] && \text{(2.6d)}
\end{aligned}$$

Secondly, a function is established to estimate the value associated with taking action a within state s under policy π. This is named as "Action Value Function" and denoted as $q_\pi(s, a)$. It can be defined as:

$$q_\pi(s, a) = \mathbb{E}\left[G_t | S_t = s, A_t = a\right] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right] \tag{2.7}$$

The goal of an agent then can be formulated with previously defined functions, which is selecting the optimum policy where it obtains the maximum cumulative expected reward. Equations 2.8a and 2.8b define the optimal state value function and optimal action-value function, respectively:

$$V_*(s) = \max_{\pi} V_\pi(s) \tag{2.8a}$$

$$q_*(s,a) = \max_{\pi} q_\pi(s,a) \tag{2.8b}$$

By combining equations 2.4 and 2.7, optimal state value function in Equation 2.8a can be rewritten:

$$\begin{aligned}
V_*(s) &= \max_{a \in A(s)} q_\pi(s,a) && \text{(2.9a)}\\
&= \max_a \mathbb{E}_{\pi_*}\left[G_t | S_t = s, A_t = a\right] && \text{(2.9b)}\\
&= \max_a \mathbb{E}_{\pi_*}\left[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a\right] && \text{(2.9c)}\\
&= \max_a \mathbb{E}_{\pi_*}\left[R_{t+1} + \gamma V_*(S_{t+1}) | S_t = s, A_t = a\right] && \text{(2.9d)}\\
&= \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V_*(s')] && \text{(2.9e)}
\end{aligned}$$

Equations 2.9d and 2.9e are Bellman optimality equations for V_*, and be later used in this section. Following the same, the Bellman optimality equation for q_* can be written, as in Equation 2.10b:

$$\begin{aligned}
q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1,a} | S_t = s, A_t = a\right] && \text{(2.10a)}\\
&= \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} q_*(s',a)\right] && \text{(2.10b)}
\end{aligned}$$

Suppose that the underlying system dynamics are entirely known (means that $p(s',r|s,a)$ function is known). In that case, one can use Dynamic Programming (Either using Policy Iteration or Value Iteration) [3] to obtain the optimal policy. If not known, one can use either Monte Carlo Based Predictions [9], Temporal Difference (TD) Learning (Such as Q-Learning or SARSA) [3] or derivations of such approaches. Monte Carlo methods approximate the value function using the average returns obtained from sampling the environment. On the other hand, TD-based methods use both Monte Carlo (uses sampling to get an average return) and Dynamic Programming (uses bootstrapping from successors) approaches. The resulting update function is given in Equation 2.11:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right] \tag{2.11}$$

Q-Learning [10] is an off-policy variant of Temporal Difference (TD) learning. It works by directly approximating the optimal action-value function. Under certain conditions, including that each state-action pair is visited infinitely often and the learning rates are suitably decreased over time, Q-learning has been proven to converge to the optimal policy as the number of iterations approaches infinity. In Q-Learning, the term 'off-policy' refers to the methodology used by Q-learning to approximate the optimal action-value function. More specifically, Q-learning estimates this function directly, without the need for the agent's actions to consistently follow the current policy. During the learning process, a Q-learning agent often employs an exploration strategy (like ϵ-greedy) to effectively improve its estimates of the Q-values. This exploration doesn't always align with the optimal policy, thereby demonstrating the off-policy nature of Q-learning. The pseudocode of the algorithm is given in Listing 1.

2.3 Partially Observable Markov Decision Processes (POMDPs)

Previously in Section 2.1, Markov Decision Processes were defined. To recall, MDPs provide a mathematical framework for modeling discrete time stochastic control/decision-making processes while having an important assumption of satisfying the Markov property. In this section, the definition of MDPs is extended in a more general form by removing the assumption of complete observability. Partially Observable Markov Decision Processes (POMDPs) are a general model for describing an environment

Listing 1 Pseudocode of the Q-Learning Algorithm. Reference: [3]

```
Algorithm parameters:
    step size $\alpha \in (0, 1]$,
    small $\epsilon > 0$,
    discount factor $\gamma$
Initialize $Q(s, a)$ for all $s \in S$, $a \in A(s)$,
    arbitrarily except that $Q(s_{terminal}, .)$=0
Loop for each episode:
    Initialize $s$
    Loop for each step of the episode:
        Choose $a$ from $s$ using policy derived from Q (e.g., $\epsilon$-greedy)
        Take action $a$, observe $r$, $s'$
        $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_a Q(s', a) - Q(s, a)\right]$
        $s \leftarrow s'$
    Until $s$ is terminal
```

with limited information.

A POMDP is defined as a six tuple $\langle S, A, T, R, \Omega, O \rangle$, Reference: [11], where:

- S is the finite set of states,
- A is the finite set of actions that the agent can take,
- $T : S \times A \times S \to [0, 1]$ is the transition function,
- $R : S \times A \times S \to \mathbb{R}$ is the reward function,
- Ω is the finite set of observations and
- $O : S \times \Omega \to [0, 1]$ is the observation function.

Contrasting with the four-tuple definition of MDPs, the definition of a POMDP extends the MDP framework by incorporating a set of observations Ω and an observation probability function O. Since the agent has no direct access to the state s, it receives an observation $o \in \Omega$ with the probability of $O(s, o) = p(o|s)$.

Table 2.1 summarizes the classification of Markov models in terms of observability and ability to control the underlying model. It's essential to note that not every stochastic process that isn't an MDP is automatically a POMDP. The foundational characteristic shared by both MDPs and POMDPs is that they are Markovian (i.e., they hold the Markov property). It's the level of observability that determines whether a process is classified as an MDP or a POMDP.

Due to the uncertainty of the observations, more than one state may correspond to the same observation. This phenomenon is known as Perceptual Aliasing [12], and since the agents cannot separate distinct states based on these observations, finding the optimal policy becomes challenging. The common misconception arises when considering the optimal control of unknown POMDPs, where traditional Reinforcement Learning (RL) approaches cannot be directly applied. The challenge here lies in the fact that these RL algorithms are generally designed under the assumption of Markov Decision Processes (MDPs), wherein full observability of the environment is guaranteed. POMDPs, however, incorporate partial observability, which makes the standard application of tools such as Bellman Equations and dynamic programming more complicated. This is not to say these methods cannot be used at all, but rather, they must be adapted to work effectively within the context of belief or hidden states in POMDPs, which will be mentioned in Section 2.4

Markov Models		**Do we have control over the state transitions?**	
		NO	**YES**
Are the states completely observable?	**YES**	**Markov Chain**	**MDP** Markov Decision Process
	NO	**HMM** Hidden Markov Model	**POMDP** Partially Observable Markov Decision Process

Table 2.1: Classification of Markovian models according to the observability and ability of controlling the underlying model. Reference: [1].

2.4 Optimal Control/Decision Making of POMDPs

When partial observability occurs, agents must make their decisions under uncertainty, and therefore, optimal decisions cannot be made at every step without forming a context or a memory. There exist two primary ways to handle the uncertainty of POMDPs in the literature [13, 14, 15, 16]. Firstly, agents can learn a sensor model where observations can be mapped into states by using the prior experiences of the agent. More formally, a probability distribution over observations given states can be used to obtain optimal policies for POMDP problems. On the other hand, instead of learning a full-sized probability distribution of observations given states (which requires knowing the transition function T of the underlying dynamics), agents can estimate their hidden state at each step by only using prior experiences (without knowing the transition function). In the subsequent subsections, these belief and hidden state POMDP definitions are mathematically defined. It is also demonstrated why defining a belief state becomes intractable in practice. Moreover, hidden state POMDPs are defined in the following part, which forms the underlying assumption of this study.

2.4.1 Belief State POMDPs

In the following, a belief state is used from [3] to create an updated POMDP, which can be optimally solved.

The probability of state s under the belief state b can be defined as $b(s)$, and an agent can update its belief state with each iteration, from literature [3], with Equation 2.12:

$$b'(s') = \eta \cdot O(o|s', a) \cdot \sum_{s \in S} T(s'|s, a) \cdot b(s) \tag{2.12}$$

Where $b'(s')$ is the updated belief state of any arbitrary state s' after taking action $a \in A$ and observing the observation $o \in \Omega$. $O(o|s', a)$ is the observation function and $T(s'|s, a)$ transition function defined in Section 2.3. In this approach, this transition function is assumed to be known. Lastly, η is a normalization constant which makes this expression a probability function, from literature [3], is defined in Equation 2.13a. η is used for normalizing the belief state so that the sum of all belief states is equal to unity, $\sum_{s' \in S} b'(s') = 1$.

$$\eta = \frac{1}{Pr(o|b,a)} \tag{2.13a}$$

$$Pr(o|b,a) = \sum_{s' \in S} O(o|s',a) \cdot \sum_{s \in S} T(s'|s,a) b(s) \tag{2.13b}$$

In theory, belief states can be further augmented with POMDP formulation to obtain an MDP where every belief is a state. A Belief MDP is defined as a four tuple $\langle B, A, \tau, R \rangle$ [17], where:

- B is the finite set of belief states,
- A is the finite set of actions that the agent can take (same as POMDP),
- $\tau : B \times A \times B \to [0, 1]$ is the transition function,
- $R : B \times A \times B \to \mathbb{R}$ is the reward function.

The belief state transition function τ and reward function over belief states now need to be defined again by using POMDP terms. In Equation 2.14a, the belief state transition function is defined. It is defined as a sum over all possible observations o, and essentially computing the total probability of transitioning from belief state b to belief state b' under action a, with consideration of all observations. The term $Pr(o|b, a)$ was defined in Equation 2.13b and the term $Pr(b'|b, a, o)$ was defined in Equation 2.14b. Lastly, the reward function $R(b, a)$ is defined in Equation 2.14c, from reference [3].

$$\tau(b'|b,a) = \sum_{o \in \Omega} Pr(b'|b,a,o) Pr(o|b,a) \tag{2.14a}$$

$$Pr(b'|b,a,o) = \begin{cases} 1, & \text{if } b' \text{ transitioned by } b, o \text{ and } a \\ 0, & \text{otherwise} \end{cases} \tag{2.14b}$$

$$R(b,a) = \sum_{s \in S} b(s) R(s,a) \tag{2.14c}$$

In a Belief MDP, each state is represented by a belief state, and the function for transitioning between states is defined by how these belief states are updated. Given an action and an observation, the transition is deterministic, as it's fully defined by the belief update rule.

The probability equation $Pr(b'|b, a, o)$ is defined in Equation 2.14b, the new belief state b' is calculated by using the belief update rule on the current belief state b, under action a and the observation o. If a proposed belief state matches the belief state obtained by applying the update rule, then Pr(b'| b, a, o) is one, otherwise, it's zero. This reflects the deterministic nature of the belief state transition function.

For the given formulation of Belief MDP, policies can then be defined with belief states. $a = \pi(b)$. The Bellman optimality equations for $V_*(b)$, under Belief MDP formulation, is defined in Equation 2.15

$$V_*(b) = \max_{a \in A(b)} \left(R(b, a) + \gamma \sum_{o \in \Omega} Pr(o|b, a) \sum_{b' \in B} \tau(b'|b, a) V_*(b') \right) \tag{2.15}$$

This Bellman optimality equation under the Belief MDP formulation simply denotes that the optimal value of a belief state b comes from the action a that maximizes the summation of immediate reward and the expected discounted optimal value of the next belief states. The expectation operator is taken for both the observations and the next belief states.

From now on, the methodologies and approaches mentioned in Section 2.2, such as Q-Learning, can be applied to solve Belief MDP problems.

It's very important to mention that, while theoretically, one can convert a POMDP to a Belief MDP, this transformation may not always be practically feasible in practice.

It is also important to mention that, belief states in POMDP problems are often high-dimensional and/or continuous, therefore working directly with these belief states are very difficult. In most cases, exact solutions are intractable due to the large state space size. An alternative approach for solving these problems is by using various approximation methods, then these problems can be tractable and optimal policies can be approximated. Even in such cases, obtaining successful policies depends on the approximated belief state design.

The general steps of designing a belief state for POMDP problems can be summarized as follows:

- **Defining the state space:**
  This step includes defining the belief state space by deciding all possible belief states the environment can be in, these states can also be discrete or even continuous.
- **Initializing the belief state:** In many POMDP cases, agents do not know any information about the environment state in initialization. In these cases, the belief state can be initialized as a uniform distribution which means that the agent believes it is equally like to be in any state. If any information exists, the initialization should be made accordingly.
- **Defining the belief update function:** The belief update function needs to be defined according to the belief state space. In a given belief state, when the agent makes an action and observes a new observation, this function defines the next belief state.
- **Updating belief state:** Lastly, as an agent interacts with the environment, the belief state must be updated accordingly in each time step. With the given belief state, the agent's current policy can be calculated.

In Listing 2, the pseudocode of the Q-Learning algorithm is updated for the Belief MDP formulation. In theory, with infinite computational resources and time, the Belief-State-Q-Learning algorithm can solve any POMDP task. Reasons of this is only possible in theory but not in practice are defined as follows:

- **Full Exploration:**
  In theory, such an algorithm must explore all belief-action tuples infinitely often to fully explore the environment and guarantee convergence.
- **Appropiate Belief State Representation:**
  The belief state must capture all of the required information about the environment, therefore representation of the belief state must be accurate for the given POMDP problem.

Listing 2 Pseudocode of the Belief-State-Q-Learning Algorithm.

```
    Algorithm parameters: step size $\alpha \in (0,1]$, small $\epsilon > 0$
    Initialize $Q(b,a)$ for all $b \in B$, $a \in A(b)$,
    Loop for each episode:
        Initialize b
        Loop for each step of the episode:
            Choose a from b using policy derived from Q (e.g., $\epsilon$-greedy)
            Take action $a$, observe $r, o'$
            Update the belief state $b'$ using observation $o$ and action $a$
            $Q(b,a) \leftarrow Q(b,a) + \alpha(r + \gamma max_{a'} Q(b',a') - Q(b,a))$
            $b \leftarrow b'$
        Until $b$ is terminal
```

- **Sufficient Discretization:**
  If the belief state space is continuous, the belief state space must be sufficiently discretized to accurately approximate the continuous belief state space and find the optimal policy.
- **Enough Computation:**
  For a single algorithm to solve any POMDP problem, infinite computational resources and time are needed to guarantee convergence.

2.4.2 Hidden State POMDPs

In practical cases, it is evident that computational limitations and the high dimensional state spaces make it impossible for a single algorithm to solve all POMDP problems. In addition, the state transition function T is unknown in most of these cases. Therefore, solving these cases requires simple and more efficient estimations that do not need to maintain a complete belief distribution, thereby being more computationally tractable but less expressive in terms of representing the original states.

Existing literature [14] classifies hidden states as follows:

- **Visible State.** These are the states which are not hidden to the agent at all times.

- **Non-Markov Hidden State.** These states cannot be directly observable to the agent, but it is correlated with the past states and/or actions, therefore called non-markov. Agents can recover these non-markov hidden states by using the memory of past observations/actions since it is correlated with the past.

- **Invisible Hidden State.** Lastly, these are the last kind of states where the given state is not correlated with anything to the agent. These kind of states are also a part of the environment, but the agents cannot make any estimates since it is independent and therefore, cannot be predicted/estimated by the agents.

Regarding to this classification [14], recoverable hidden states can be estimated to optimally solve some POMDP tasks. Approaches to solve/estimate states are initially divided into two classes, namely the Memoryless(Stateless) and Memory-Based approaches.

Memoryless agents that are using reinforcement learning with hidden state POMDPs do not try to disambiguate these aliased observations. Instead, it simply ignores hidden state problems and applies traditional reinforcement learning methods like there is no perceptual aliasing. These approaches try to obtain the best policy without additional memory or context, which performs poorly in most partially observable problems. It has also been proven that finding the optimal memoryless policy in Hidden State POMDP problems is NP-Hard [18].

Contrary to the first approach, memory-based approaches use an additional source of memory (or context) to estimate the underlying state by combining this additional information with given observations. In the literature, various strategies are used to represent these additional sources. These include employing a fixed-length window of recent observations, finite state machines, external memories (which can store observations, actions, rewards, or any blend of such data), and recurrent neural networks. Approaches based on recurrent neural networks can also be termed as "internal memories" since the memory data is preserved within the agent's policy itself. Since the way of representing memories changes the formal definition, in a more abstract way, a Hidden State MDP can be defined as a four tuple $\langle S', A, \tau, R \rangle$ where:

- S' is the finite set of context/hidden states,

- A is the finite set of actions that the agent can take (same as POMDP),
- $\tau : S' \times A \times S' \rightarrow [0, 1]$ is the transition function,
- $R : S' \times A \times S' \rightarrow \mathbb{R}$ is the reward function.

Similar to the Belief State MDPs, Reinforcement Learning based approaches can then be used to learn the optimal policy. The resulting pseudo-code of this algorithm is the same as Listing 2, except that context/hidden states $s' \in S'$ are used instead of belief states in the notation.

Lastly, it's worth mentioning when augmenting the environment with a hidden state using practical memory representations, issues such as computational complexity, discretization, and the curse of dimensionality can prevent these augmented environments from being treated as true MDPs. Furthermore, these practical issues can make it challenging to obtain the exact global policy or some approximated policies (but yet, may still be achievable). One concrete example of this expression, for memory-based representations, it has been proven that [19] even though a globally optimal policy exists for a memory-augmented environment, the memory-augmented environment itself might not be an MDP. For these cases in which the augmented environment is not MDP, and this non-Markovian case impacts the performance of RL agents that explicitly exploit the Markovian assumption, learning may become unstable. Therefore, it is crucial to efficiently design hidden states and use problem-specific adaptations.

2.5 Neural Network Architectures

Artificial neural networks can be described as computational models that are inspired by the neural structure of the brain. These networks have the capacity to learn complex functions from data. There are several studies that trace the origins of neural networks back to the early to mid-20th century [20, 21], drawing influences from areas such as biology, neuroscience, psychology, and computer science. To the best of our knowledge, the earliest prototypes of artificial neural networks date back to the study by McCulloch and Pitts in 1943 [22]. A neural network can be characterized by its linear and/or nonlinear processing units, often referred to as layers, and by the employment of backpropagation, which leverages the chain rule to concurrently adjust the parameters of the entire network at once.

In literature, a variety of neural network architectures have been proposed, with each with its unique features, advantages, and trade-offs. In this section, neural network architectures are characterized and defined based on their structure. Subsequently, concepts utilized in this study are defined to provide background information to the readers.

- **Multi-Layer Perceptrons and Convolutional Neural Networks:**

  Multi-layer perceptrons (MLPs), are the very simplest form of artificial neural networks and consist of multiple layers, in which at each layer, multiple neurons exist, and each neuron is connected to every neuron in the next layer, constructing a dense network in terms of the number of connections.
  A feedforward neural network chains together multiple function compositions, and thus it can be referred to as a 'network'. This model can also be represented as a directed acyclic graph (DAG), where each node corresponds to a function in the sequence, highlighting the layered, sequential structure of these networks [23]. This chain form can be denoted as $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$. In this notation, $f^{(1)}$ is the first layer of the network, and so on. A linear layer of a neural network can be defined as $\hat{y} = W^T h + b$, where $\hat{y}$ is the predicted/calculated output value/label, W is the weight matrix which provides the weights of the linear transformation, h is the input features of the current layer and lastly, b is the bias values. In a more compact representation, For a given fully connected layer l, if h^l represents the output of that layer (also the input to the next layer $l + 1$), then the output $h^{(l+1)}$ of the next layer can be calculated as:

  $$h^{l+1} = \sigma(W^{l+1} h^l + b^{l+1})$$

  Where:

  - W^{l+1} is the weight matrix for layer $l + 1$,
  - $b^{(l+1)}$ is the bias vector for layer $l + 1$.

- σ is the activation function. (Mostly used functions of σ are ReLU (Rectified Linear Unit), sigmoid, or tanh functions) .

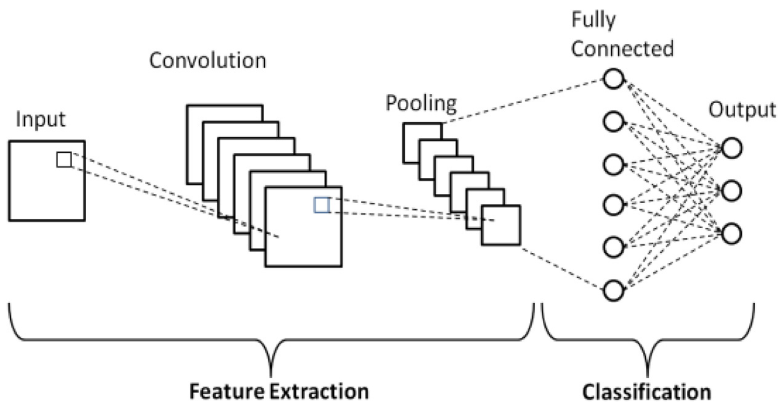


Figure 2.3: Example structure of a simple convolutional neural network followed by an MLP, Source: [4].

Convolutional NNs utilize the convolution [24] operator to capture spatial relationships in data when jointly used with fully connected neural networks. It allows the representation of spatial information in a compressed form, and with this architectural improvement, the number of parameters in a neural network can be decreased, and the chance of overfitting can be reduced when compared to usingly only MLPs. An example illustration is shared in Figure 2.3.
For a 2D input x (which can be an image), a 2D convolution operation can be defined as follows:
$y_{i,j} = (x * w)_{i,j} = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} x_{i+m,j+n} w_{m,n}$
Where:

- w is a 2D kernel of size $k \times k$,
- i and j are the spatial coordinates in the output y,
- $w_{m,n}$ is the kernel weights of the given spatial coordinates, and these weights are shared across the entire spatial extent of the input, which is a key aspect of the CNN that allows it to learn spatial relationships.

The convolution operation can be extended to 3D inputs. Typically, this layer is used in conjunction with a pooling layer, which performs downsampling on its inputs and is often followed by fully connected layers in the network architecture.

- **Recurrent Neural Networks (RNNs):**

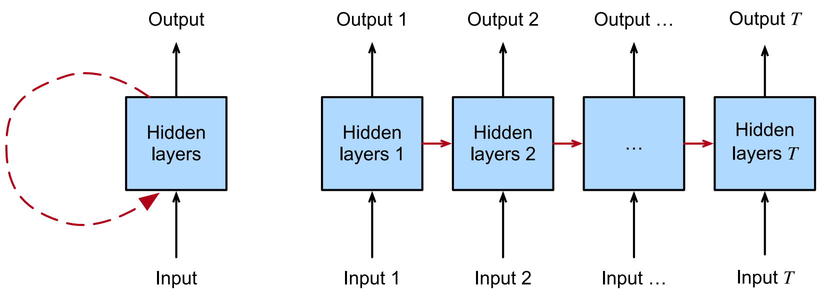


Figure 2.4: Example single neuron of a basic recurrent neural network. On the left, recurrent connections are drawn with cycles, and on the right, the recurrent neuron is unfolded over time steps. Source: [5].

Recurrent Neural Networks (RNNs) utilize an internal memory structure to process sequences of inputs, enabling the preservation of sequential information within the network structure. To the best of our knowledge, Long Short-Term Memory (LSTM) networks [25] and Gated Recurrent Unit (GRU) networks [26] are among the most widely recognized architectures for the practical implementation of recurrent neural networks. Example illustration is shared in Figure 2.4.
In most simple terms, a recurrent neural network layer can be defined as follows:

Given:

- An input sequence $\mathbf{x} = (x_1, x_2, ..., x_T)$,
- A hidden state h initialized often as zeros,
- Weight matrices W_{xh}, W_{hh}, and W_{hy},
- Bias vectors b_h and b_y,
- An activation function, typically the hyperbolic tangent $\tanh$,

$$h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$

$$y_t = W_{hy}h_t + b_y$$

for each $t = 1, ..., T$.

LSTM (Long Short-Term Memory) networks are a special type of RNNs (Recurrent Neural Networks) designed specifically to avoid the long-term dependency problem, enabling them to remember and learn from long sequences without suffering from the vanishing gradient issue prevalent in traditional RNNs. An LSTM layer is defined by the following equations from [27]:

$$\begin{aligned}
\text{Forget Gate:} & \\
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
\text{Input Gate:} & \\
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
\tilde{c}_t &= \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \\
\text{Update Cell State:} & \\
c_t &= f_t * c_{t-1} + i_t * \tilde{c}_t \\
\text{Output Gate:} & \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
h_t &= o_t * \tanh(c_t) \\
\text{Overall Representation:} & \\
(o_t, h_t, c_t) &= \text{LSTM}(x_t, h_{t-1}, c_{t-1})
\end{aligned}$$

Where:

- σ denotes the sigmoid function.
- $\tanh$ is the hyperbolic tangent function.
- $[h_{t-1}, x_t]$ means the vectors h_{t-1} and x_t are concatenated.
- $*$ denotes element-wise multiplication.

- **Autoencoder Networks:**

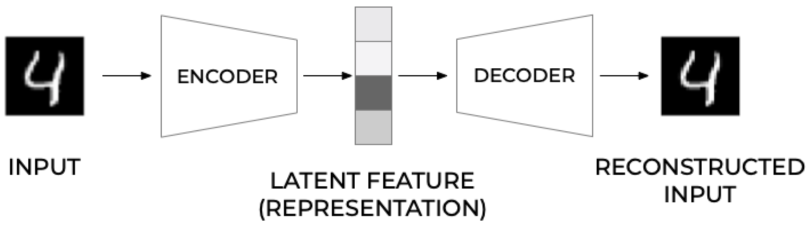


Figure 2.5: Example structure of a basic autoencoder network, Source: [6].

An autoencoder network [28] is a specialized network for utilizing unsupervised learning where it learns to compress the input data (o) and subsequently attempts to reconstruct it, denoted as $\hat{o}$, solely based on the compressed latent representation z [29]. An example illustration is shared in Figure 2.5.

Encoder:

$$z = Encoder(o)$$

Decoder:

$$\hat{o} = Decoder(z)$$

Overall Representation Of The Autoencoder:

$$\hat{o} = Autoencoder(o) = Decoder(Encoder(o))$$

Moreover, different architectures also have different computational requirements. Some might be more memory-intensive, while others might be more computationally demanding. The decision to choose an architecture also depends on these factors, especially when dealing with large-scale problems or hardware constraints. These architectures can also be combined with approaches in Section 2.6 to handle many challenges, such as dealing with visual inputs or learning sequential information in Reinforcement Learning tasks.

2.6 Deep Reinforcement Learning

In section 2.2, the concept of Reinforcement Learning was defined, and several algorithms for solving RL problems were mentioned, including the Q-Learning algorithm. However, in practice, only a small set of problem domains can be efficiently solved by using these traditional approaches due to these algorithms tends to be computationally intractable as the state and action spaces grow large or become continuous.

Deep Reinforcement Learning (Deep RL) overcomes this limitation by using deep neural networks (previously defined in section 2.5) as function approximators to estimate either a policy, a value function, or both. In most simple terms, Deep RL can be defined as the integration of deep learning methodologies into reinforcement learning problems to eliminate possible drawbacks in existing solutions. This approach enables the handling of large or even continuous state and action spaces, which were previously intractable.

However, the use of deep networks as function approximators introduces an approximation error. This error arises when the approximated function does not perfectly represents the true function. Therefore, even if all state-action pairs are visited infinitely often (a requirement for convergence guarantee in traditional reinforcement learning), the agent may not converge to the optimal policy due to this approximation error. In practice, several approaches [30, 31, 32, 33] can be adapted to improve this trade-off condition but it cannot be fully avoided.

In Listing 3, the pseudocode of the "Deep Q Learning (DQN) [30]" algorithm is given. The main difference from the traditional "Q-Learning" algorithm is that the action-value function Q is now approximated by a deep neural network parameterized with θ. To recall, the traditional "Q Learning" is considered an off-policy method because its Q-value update rule uses the greedy policy (best possible action) to update its Q-values, while the behavior policy that is actually followed during learning can be non-greedy (like ϵ-greedy) or a different exploratory policy. Similar to traditional Q Learning, the DQN algorithm is also considered an off-policy method due to a similar greedy update rule, also combined with the experience replay. Transitions (s_j, a_j, r_j, s_{j+1}) are sampled from a replay memory $\mathcal{D}$ (with size N) to optimize the neural network with mini-batches.

Listing 3 Pseudocode of the Deep Q-Learning Algorithm, Reference: [30].

```
Algorithm parameters:
    step size $\alpha \in (0, 1]$,
    small $\epsilon > 0$,
    discount factor $\gamma$,
    capacity of replay memory $N$
Initialize replay memory $\mathcal{D}$ with capacity $N$
Initialize action-value neural network function $Q$
    with randomly initialized weights $\theta$
Loop for each episode:
    Initialize $s$
    Loop for each step of the episode:
        Choose $a$ from $s$ using policy derived from Q (e.g., $\epsilon$-greedy)
        Take action $a$, observe $r$, $s'$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(s_j, a_j, r_j, s_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } s_{j+1} \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta) & \text{for non-terminal } s_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$ with respect
        to neural network parameters and update $\theta$
        $s \leftarrow s'$
    Until $s$ is terminal
```

@articlekloek1978bayesian, title=Bayesian estimates of equation system parameters: an application of integration by Monte Carlo, author=Kloek, Teun and Van Dijk, Herman K, journal=Econometrica: Journal of the Econometric Society, pages=1–19, year=1978, publisher=JSTOR

2.6.1 PPO

Deep Q-Learning (DQN) aims to estimate the action-value function by minimizing the difference between the estimated Q-values and the target Q-values. Once this function is learned, a deterministic policy can be derived by selecting the action that maximizes the Q-value for a given state.

In contrast, Proximal Policy Optimization (PPO) [34] is one of the most preferred choices in policy gradient/optimization methods aiming to address the challenges of policy gradient methods related to large policy updates. Instead of learning a value function, it directly optimizes a parameterized policy to maximize expected rewards.

Traditional policy gradient methods allow policies to be updated such that they can become significantly different from the original. This can sometimes lead to destabilized training or catastrophic drops in performance. PPO algorithm proposes a surrogate objective function to bound the deviation of the new policy from the old one. This is achieved by introducing a constraint or penalty to the optimization objective, promoting incremental policy updates. Such a modification offers enhanced stability and reliability in comparison to conventional policy gradient methods. Mathematically, PPO achieves this enhancement by adjusting the policy gradient objective to incorporate a clipped version of the surrogate objective function. The idea is to avoid making excessively large policy updates by clipping the objective to lie within a certain range of the original policy. Specifically, PPO minimizes the following objective in Equation 2.16:

$$L(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{2.16}$$

Where:

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio between the new policy and the old policy. If $r_t(\theta) > 1$, the action a_t at state s_t is more likely in the current policy than the old policy. And if $0 < r_t(\theta) < 1$, the action is less likely for the current policy than for the old one. This acts as an estimate of the divergence between the old and current policy.

- $\hat{A}_t$ is an estimator of the advantage function at time t.
- ϵ is a hyperparameter that controls the extent to which the policy can be updated in one step.

By clipping the policy update, PPO ensures that the new policy doesn't deviate significantly from the old policy, maintaining stable and robust learning.

In summary, while Deep Q-Learning (DQN) focuses on estimating the action-value function and then derives a deterministic policy from it by selecting the action that maximizes the Q-value for a given state, Proximal Policy Optimization (PPO) takes a different approach which directly optimizes a stochastic policy to maximize expected rewards. Another distinguishing feature is in their experience management: DQN uses an experience replay mechanism to stabilize its learning, leveraging past experiences to train the model. On the other hand, PPO operates on policy, relying on the experiences collected from the current policy's interactions with the environment. PPO estimates the policy gradient using multiple trajectories from these experiences. This difference in experience management also categorizes DQN as an off-policy algorithm and PPO as an on-policy algorithm.

2.7 Intrinsic Motivations

The term intrinsic motivation comes from the area of psychology, where it is defined as the act of engaging in some activity mostly for curiosity, satisfaction, and fulfillment, not expecting any extrinsic reward or an outcome [35]. Similarly, this concept has been adapted to the fields of artificial intelligence and robotics, and it can be applied to agents where it seeks curiosity instead of pursuing external rewards. While the concept of intrinsic motivation often overlaps with the idea of exploration in reinforcement learning, it provides a broader framework than just exploration. Intrinsic motivation is a more comprehensive concept that drives the agent's actions not only by curiosity or the need to uncover unknown parts of the environment (exploration) but also by other internal factors that can drive learning and behavior. Intrinsic motivations in reinforcement learning can be classified into two main topics [36]:

- **Knowledge Acquisition:**
  Within this area, the aim of intrinsic motivation for agents is to find new knowledge for the given environment. Knowledge Acquisition can be further classified into three topics, such as Exploration (motivating agents to improve exploration in sparse environments), Empowerment (motivating agents to states where it has maximum control over the environment), and lastly, State Representation (motivating agents to learn proper and related state representations).

- **Skill Learning:**
  Intrinsic motivations can encourage and motivate agents to construct and learn task-independent skills where these skills are useful for many problems and particular skills could be reused to help agents quickly adapt to new problems.

Lastly, it is notable that in the current literature, studies employing intrinsic motivations in the context of POMDP problems are relatively sparse compared to those focused on MDP problems [36]. Furthermore, the application of intrinsic motivations in these complex contexts might introduce additional layers of difficulty. Despite these challenges, the potential gains of intrinsic motivation in encouraging exploration, learning from sparse rewards, and motivating long-term planning make this an important aspect for further research.

2.8 Related Works

Approaches With Internal Memory Mechanisms in POMDP environments. In previous works, k-order memories are used to form a memory for the most recent observations in order to solve visual atari problems [37] and showed above human-level performance in such scenarios. The U-Tree method [38] uses a variable history window to represent long-term dependencies. Similarly, [39] uses the history window approach for reinforcement learning with hidden states. [40] uses RNNs in order to approximate the value function. The state of the environment is approximated by the current observation and the recurrent activations of the neural network, which represent the agent history. Recurrent neural networks are also directly used for solving pomdp problems in [41]. This study shows by combining LSTM networks with the Deep Q Learning method. The resulting agent is capable of detecting relevant information given prior. [42] proposes internal memory architectures by using LSTMs for high-dimensional visual observation problems.

External Memory Approaches. The study [16] begins with, wherein external binary memories are created for POMDP RL problems. In [43], continuous-valued external memories are introduced for solving continuous control tasks that necessitate memory. Unfortunately, the policy is trained with imitation learning since the reward signal was insufficient for agents to understand how memory can be controlled. In [19], authors used external memory-augmented environments to solve POMDP problems requiring long-term memory dependencies and compared external memory approaches with LSTM-based internal memory methods. Neural Turing Machines (NTMs) are a special kind of Turing architecture introduced in [44], which creates a computationally universal model for neural networks, allowing them to have a differentiable external memory matrix and a controller to read/write memory contents, which can be trained end-to-end, with supervised learning. Following this approach, [45] combines NTMs with reinforcement learning to move the memory tape head for read and write operations, which can be used for external memory. One similar approach to NTMs is the Neural Map [46], which is different from NTMs, where the external memory is structured as a spatial 2D grid or map. Self Memory Management from [47] utilizes an external memory to form a context for a given POMDP problem. Several other works exist in the literature where the problem domain changes (continuous control and transport optimization) [48, 49] while still using similar external memory approaches.

Count-Based Intrinsic Motivation. These works leverage counting or statistical frequency techniques to derive intrinsic motivation, encouraging agents to explore states or actions that are less frequently visited or known. Jo et al. propose a learnable hash-based episodic count for complex problems [50]. Machado et al. introduce the substochastic successor representation as a count-based exploration mechanism [51]. Martin et al. delve into exploration within feature spaces using count-based strategies [52]. Ostrovski et al. propose a count-based exploration strategy using neural density models to gauge the density of states [53]. Bellemare et al. combine both count-based exploration and intrinsic motivation methodologies into a unified framework [54]. Tang et al. conducted an in-depth study on count-based exploration in the realm of deep reinforcement learning [55]. Zhang et al. delve into the idea of pushing exploration boundaries through the "Bebold" methodology, seeking novel states over the inverse state visitation counts [56].

Novelty-Based Intrinsic Motivation. Works in this category emphasize the detection and exploration of novel experiences, states, or actions in the environment, leveraging the agent's intrinsic motivation to seek out the unknown. Burda et al. focus on novelty-based exploration through random network distillation, emphasizing the importance of exploring novel states [57]. Kubovcik et al. approach novelty from a signal detection perspective, especially in the context of robotic applications [58]. Pathak et al. employ a self-supervised prediction approach, driving the agent's exploration with intrinsic curiosity towards unfamiliar situations [59]. Seurin et al. emphasizes the utility of actions in the exploration, advocating for actions that lead to novel experiences or learnings [60].

CHAPTER 3

SOLVING PARTIALLY OBSERVABLE VISUAL REINFORCEMENT LEARNING TASKS

In previous chapters, a comprehensive exploration of several fundamental concepts has been conducted. Equipped with this foundation, our attention now shifts to a more intricate and practical problem domain - solving of the Partially Observable Visual Reinforcement Learning Tasks. This chapter will bring together the concepts we've previously discussed, illuminating the challenging terrain of visual environments where state information is not always fully available or directly observable. The exceptional hurdles presented by this scenario will be analyzed, and an exploration of how deep learning can be leveraged to attain optimal behaviors from these tasks will be undertaken.

Contrary to MDPs, which are fully observable and for which optimal solutions can often be computed given sufficient resources, solving POMDPs poses a greater challenge due to their inherent partial observability. Although using computational methods can significantly improve our chances of solving problems known as POMDPs, it does not guarantee a solution to every POMDP problem. As previously mentioned, deriving optimal solutions for POMDP problems is typically computationally intensive, and may even be intractable in some cases. Furthermore, there is no universally 'perfect' algorithm or approach that can solve all POMDP problems, reflecting the diversity and complexity of these environments.

In this chapter, the formulation of approaches specifically aimed at addressing a selected subset of POMDP problems that can be effectively solved using appropriate methods is presented. To be more precise, this thesis focuses on a specific subset of POMDP problems, that requires long-term remembering of observations to obtain optimal policies. This can be further explained by a motivating example.

3.1 Motivating Example

The academic study conducted by Bakker in 2001 [61] served as an inspiration and provided motivation for this study. Imagine a robot operating in a simple, single-corridor maze environment that consists of only a corridor and a T-shaped junction (which leads to two rooms) at the end. The robot is initially positioned at the other end of this maze. Suppose that this robot can move in cardinal directions (north, south, east, and west) in the maze to navigate. In this maze environment, one of these rooms includes a battery charger (a reward), while in the other room, the robot battery depletes (a penalty). And, suppose that the goal of this robot is to access the room with the reward inside.

The robot must initially choose a direction in t-junction at random because which room contains the reward or the punishment is initially unknown. But there's a twist: the state of the maze changes after each episode/trial, meaning that the position of the reward and punishment rooms switches randomly, and the agent does not know which room is which.

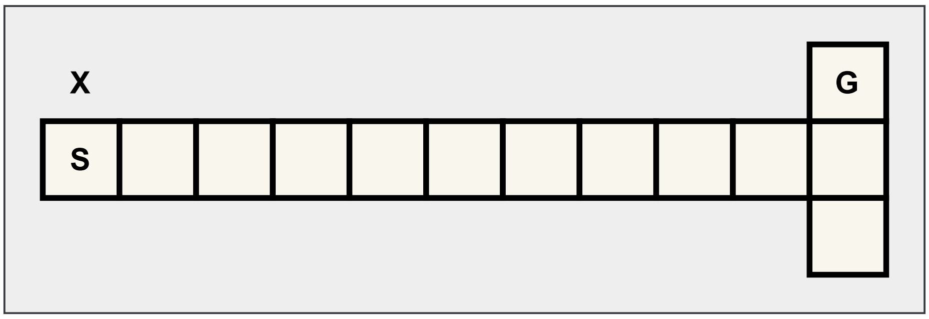


Figure 3.1: An example illustration of the `T-Maze Environment` maze with a corridor length of 10, particularly demonstrating the long-term memory dependency. The character S and G denote the start and goal states, accordingly. The position of the state G randomly changes as either the north or south state in the T-junction. The X sign in the figure denotes that a hint is available in this state.

However, there's a hint: a sign appears at the start of the maze, indicating a direction where the reward room is in the current episode. According to the scenario/definition, the range of robot vision can be either unlimited or limited to nearby state/states.

Consider a privileged robot that can navigate and has unlimited vision, which can continuously see the sign information, even when it reaches the T-junction state. In order to make optimal decisions at any state, this robot only needs its current state. In this case, this environment can be modeled as a Markov Decision Process, and an optimal policy can be obtained with the mentioned approaches in Section 2.2. Since the state space and possible actions are limited and small, obtaining the optimal policy is trivial.

Now, let us postulate a scenario when the robot's vision is limited or even consider that the robot only sees nearby features. In this case, this T-Maze environment becomes a Partially Observable Markov Decision Process (POMDP) since when the agent is in the t-junction, in order to make the optimal decision, the agent needs to remember the sign information, which was only visible at the start of the episode. Even though the underlying environment dynamics is MDP, due to the partial observability, Reinforcement Learning cannot be directly applied to solve this task. Instead, this problem needs to be addressed with a POMDP model, and appropriate methods need to be applied. This environment is a simple yet effective example of such partial observability that makes directly applied traditional methods (mentioned in Section 2.2) ineffective and even unsuccessful.

This environment, namely the `T-Maze Environment`, was first introduced by [61] to demonstrate a special kind of POMDP problem, which is handling long-term dependencies between relevant events. Figure 3.1, illustrates a grid-based visualization of the T-Maze environment with a corridor length of 10. In the notation, the scenario featuring the privileged robot with unrestricted vision is referred to as the `Fully Observable T-Maze Environment`, while the other scenario is named as the `Partially Observable T-Maze Environment`.

In practice, this environment can be implemented by either directly giving necessary information, such that a state is a three-tuple ()

Later in this chapter, proposed methods will be examined. The objective of this method is to achieve optimal or nearly optimal solutions for visual POMDP tasks that requires long-term remembering of observations. Simultaneously, the method aims to enhance efficiency in terms of the necessary quantity of observation samples needed to learn the optimal policy.

To optimally solve such POMDP problems, one can employ the methodologies outlined in Section 2.2. However, the high-dimensional state space and dependency on remembering relevant states for long terms requires special attention. When addressing these concepts with practical problems, the design of intelligent and effective methodologies is crucial and necessary. Without such methods, these environments could quickly become intractable. The proposed approaches in this chapter address these issues and aim to obtain efficient solutions to such problems.

3.2 Augmenting External Observation Memory For Visual Tasks

In this section, the focus is shifted to the definition of an external observation memory that will be utilized by agents in the subsequent sections, particularly in section 3.3 and beyond. Previously, the POMDP framework and approaches to optimally solve POMDP problems have been defined (particularly in Sections 2.3 and 2.4).

Let O represent the observation space, which is typically high-dimensional in visual RL tasks. $o_t \in O$ is the observation and a_t is the corresponding action taken by the agent at time step t. Suppose that these visual tasks are partially observable and the underlying state transition dynamics are not given (the agent does not know). Due to visual occlusions in observations, a form of memory or context is needed to estimate the hidden state, and if proper representations are used, this POMDP problem can be optimally solved.

An external observation memory, denoted as M at time step t, is a dynamic construct that empowers the agent to systematically store and retrieve observations. This structure is defined as follows:

$$m_t =< o_i, ..., o_j, o_k >$$

Where o_i, o_j and o_k represent arbitrary observations that are stored by the agent itself.

The order of the indexes i, j, and k can also be arbitrary. Although, in order to be compatible with Section 3.3, Visual Self-Memory Management, indexes are defined as $t > i > j > k$, where the most recent (lastly observed) observation is stored in the most left element in the memory.

3.3 Visual Self Memory Management

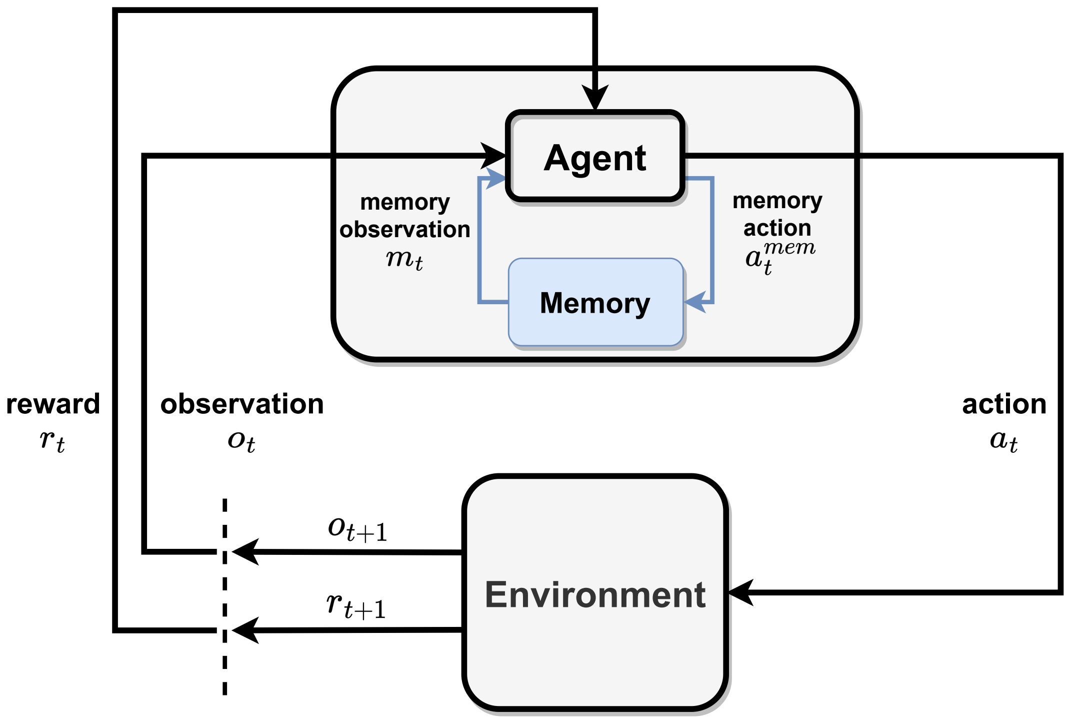


Figure 3.2: Augmentation of the external memory for a reinforcement learning agent formulation, previously illustrated in Figure 2.1. External memory related elements are colored in blue.

In social sciences, there exists a concept known as Stigmergy, which is often observed, especially in social creatures. Stigmergy refers to the fact that agents could change/alter their world (environment) to affect their future behaviors, usually in a useful way [62, 16].

From this perspective, in reinforcement learning, if agents are equipped with external memory, they can benefit themselves by interacting to learn to use a memory. As a result, this augmentation may result in converting the POMDP problem into a hidden-state MDP. Following a similar approach as seen in [16, 19, 47], the concept of external memories is extended for the establishment of a hidden-state MDP. This concept is subsequently examined in-depth, encompassing its diverse variations in terms of memory augmentation and action design. This study augments the Self Memory Management from [47] to be compatible with visual observations, and customized intrinsic motivation functions are designed to work with high dimensional/continuous observations. The term Visual Self Memory Management (VSMM) is used when dealing with visual and high-dimensional inputs. Also, in experimental results, by Section 4.2, the use of Visual Self Memory Management with Deep Reinforcement Learning is further defined.

The contents of an external memory may depend on the problem and memory requirements itself. This study augments an external memory only of observations, which is enough and sufficient in terms of the targeted POMDP problem.

An External Memory Augmented Hidden State MDP can be defined as a four tuple $\langle S', A, \tau, R \rangle$ where:

- S' is the finite set of context states (observations augmented with external memory). $S' = O \times M$, where O is the finite set of observations and M is the finite set of memories defined in 3.2
- $\bar{A}$ is the finite set of combined actions that the agent can take. $\bar{A} = A \times A^{mem}$, where A are actions defined from POMDP definition and $A^{mem} \in \{a^{push}, a^{skip}\}$ are the memory actions,
- $\bar{\tau} : S' \times A \times S' \to [0, 1]$ is the combined transition function $\bar{\tau} = \langle \tau, \tau^{mem} \rangle$, where τ is the original transition function and τ^{mem} is the memory transition function, which is defined in Equations 3.1a and 3.1b,
- $R : S' \times A \times S' \to \mathbb{R}$ is the reward function.

In Equations 3.1a and 3.1b, the memory transition function is defined. This definition originally comes from [47], and the main difference between this notation and the original definition is, in this notation, the observation o_t appending order into the memory is left-sided, where the most recent (lastly observed) observation is stored in the most left element in the memory.

$$m_{t+1} = \tau^{mem}(m_t, o_t, a_t^{mem}) \tag{3.1a}$$

$$= \begin{cases} \langle o_i, ..., o_j, o_k \rangle, & \text{if } a_t^{mem} = a^{skip}, \\ \langle o_t, o_i, ..., o_j, o_k \rangle, & \text{if } a_t^{mem} = a^{push} \text{ and } |m_t| < c, \\ \langle o_t, o_i, ..., o_j \rangle, & \text{if } a_t^{mem} = a^{push} \text{ and } |m_t| = c, \end{cases} \tag{3.1b}$$

In Figure 3.2, the external memory augmented agent is visually illustrated. As can be seen, an agent has been equipped with external memory, and it has the ability to manipulate the contents by memory actions a_t^{mem}. As a result, agents can estimate the hidden state from a given state and memory observations. When looked at from the outside of the agent, the agent is still compatible with the traditional RL framework where it receives observations, rewards, and decides actions. Therefore any RL algorithm can be used for optimization, with the exception that the RL algorithm now uses the agent's extended observation and action space. The memory-related observations and actions stay inside the agent, and only the environment-related action will be given to the environment.

3.4 Visual Intrinsic Motivations

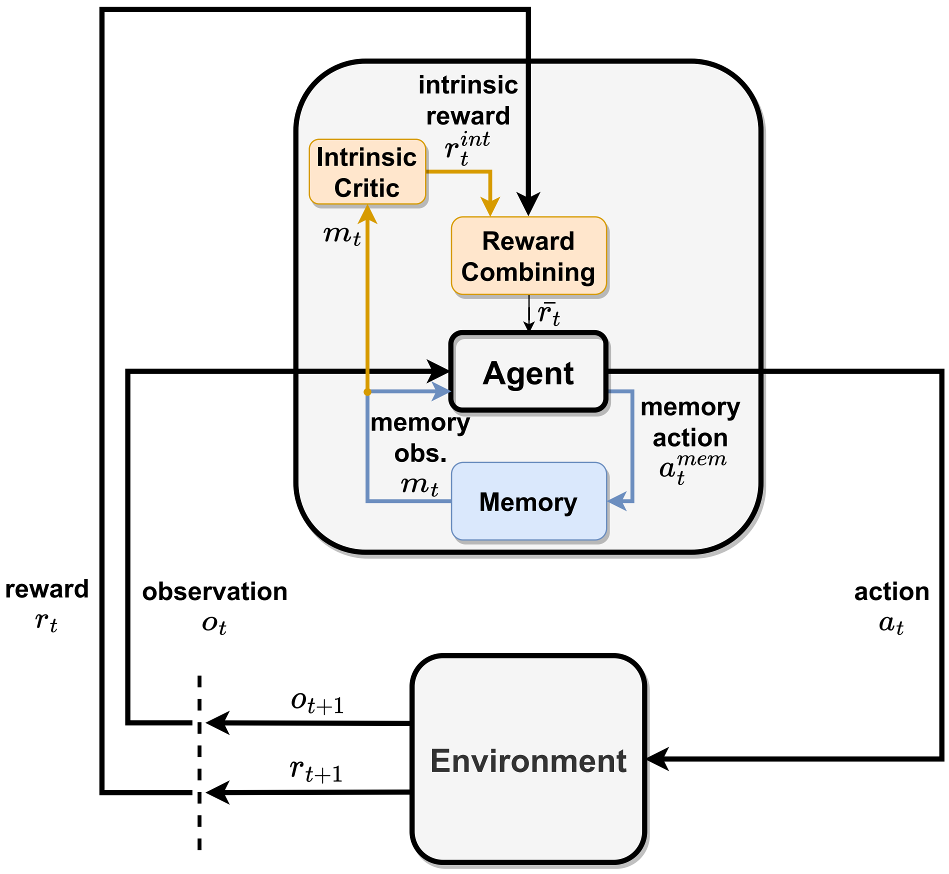


Figure 3.3: Extention of the Memory Augmented Agent Framework (Figure 3.2) with intrinsic motivation aproaches. Intrinsic motivation related elements are colored in orange.

As defined in Section 2.7, intrinsic motivations can be used for improving skill learning and encouraging exploration. From the perspective of the agent, the VSMM approach expands both the action and observation spaces by defining an external memory and the actions required for its manipulation. It is also crucial to note that, the agent faces an increased complexity in exploration due to the curse of dimensionality, which comes from extending the action and the observation space. Moreover, learning how to effectively manipulate a memory is a challenging task since there is no direct external reward to guide how to efficiently use the memory.

Similar to the study outlined in [47], the application of intrinsic motivation proves to be highly beneficial in these cases. If properly adapted, the disadvantages of the VSMM method can be minimized, and VSMM can compete with the state-of-the-art methods and even outcomes in terms of some critical performance metrics. However, designing and adapting intrinsic motivations for especially high-dimensional visual input (or even continuous value observations) is not trivial in most cases. Through this section, existing approaches are expanded upon, and effective, generalizable IM functions are developed for visual domains. These functions will be referred to as Visual Intrinsic Motivations.

In figure 3.3, the memory-augmented agent framework has been extended with intrinsic motivation approaches. As given in the figure, the intrinsic reward is calculated by a Intrinsic Critic, followed by a Reward Combining function. These two functions (denoted as R^{int} and $\bar{R}$, respectively) calculate the intrinsic reward r_t^{int} and adjust the scalar proportion with the given external reward r_t. In short, these functions will be designed in Sections 3.4.1 and 3.4.2 in two alternative and yet effective ways.

3.4.1 Efficient Count-Based Intrinsic Motivation with Perceptual Hashing

The general idea of count-based intrinsic motivations can be defined as leading agents to novel states that are not usually visited. This concept can be combined with VSMM to motivate agents to keep novel observations in their memory. Given that the type of POMDP problems being focused on requires long-term remembering of observations to obtain optimal policies, these types of POMDP problems also hold the assumption argument from [47] that assumes important observations are rare in uncertain environments. If this assumption is correct for these POMDP problems, then motivating agents to remember novel observations can improve learning performance. On the contrary, if this assumption is not correct and important observations are common, then a memory or a context might not be necessary, and more trivial approaches could be used to solve such POMDP problems (such as stacking the observations for the last n time steps).

In this section, a count-based intrinsic motivation function is extended to be used in visual problems where the observation space is discrete but high-dimensional, and effective calculation of observation counts is required to make the algorithm be tractable in these problems.

By following the assumption mentioned above, the novelty of an observation can be determined by the observation occurrence frequency $f_t(o)$, over all time steps, including all the episodes. In Equation 3.2a, the normalized frequency of the given observation is defined, and by combining these frequencies across the memory elements, a count-based intrinsic motivation function is defined in Equation 3.2b. However, given the constantly changing frequencies, the scaling of the intrinsic reward with the extrinsic reward presents a challenge without adaptive reward scaling. Instead of this direct scaling, the intrinsic reward can be dynamically normalized by using a running mean (as described in Equation 3.2c) and variance (referenced in Equation 3.2d). This normalization scales intrinsic rewards into a zero mean and unit variance distribution. Based on this normalization (shown in 3.2e), negative normalized values indicate that the original value is below the running mean. A value of zero signifies equivalence to the average, and a positive value indicates that it exceeds the running mean. In order to positively motivate agents to retain novel observations in memory, the normalized intrinsic reward values are clipped to fall within the range of [0, 1]. The total combined reward is defined in Equation 3.2f.

$$f_t(o) = \frac{n_t(o)}{\sum\limits_{o' \in \hat{O}} n_t(o')} \tag{3.2a}$$

$$R^{int}(m_t) = \left(\sum_{o \in m_t} (1 - f_t(o)) \right) - c \tag{3.2b}$$

$$\mu_t = \lambda \mu_{t-1} + (1 - \lambda) R^{int}(m_t) \tag{3.2c}$$

$$\sigma_t^2 = \lambda \sigma_{t-1}^2 + (1 - \lambda)(R^{int}(m_t) - \mu_t)^2 \tag{3.2d}$$

$$\widetilde{R}^{int}(m_t) = clip(0, \frac{R^{int}(m_t) - \mu_t}{\sigma_t}, 1) \tag{3.2e}$$

$$\bar{R}_t = R_t + \beta \widetilde{R}^{int}(m_t) \tag{3.2f}$$

Where:

- $\widetilde{R}^{int}(m_t)$ is the zero-mean normalized and clipped intrinsic reward, values in the range: $[0, 1]$. If this value is greater than zero, it means the intrinsic reward is greater than the average. The act of clipping ensures that agents are positively motivated for intrinsic rewards that are higher than the average value.
- μ_t and σ_t are the running average and variance values at time t.
- $\lambda \in [0, 1]$ is the running average adaptation hyperparameter.
- $c >= 0$ is the memory capacity, which sets the number of observations that can be stored in a memory.
- $f_t(o) \in [0, 1]$ is the normalized frequency of the observation o in time t.
- $n_t(o)$ is the total occurrence count of observation o occurs.
- $\hat{O}$ is the set of observations that have been observed by the agent. Eventually, $\hat{O} \to O$ when $t \to \infty$ with a explorative agent.
- $\beta \in [0, 1]$ is the adjusting parameter that changes the scale of the intrinsic reward.

For visual problems, each pixel is a three-valued (RGB), bounded integer in the range of 0-255. Therefore, visual problems are discrete problems with a large observation space. In data structures, counting occurrences of a large number of arbitrary values is most effectively achieved by hash maps, since it provides constant time (O(1)) operations for adding and retrieving elements. Since the observations are high-dimensional and large arrays of pixels, calculating a hash value for the given observation also takes time. One of the simplest hashing can be done by converting this array into a String and using this String as a hash value, which is time-consuming when compared to other calculations.

For the efficient hashing of observations, the utilization of Perceptual Hashing [64, 63] is proposed in this study. This technique, drawn from the realm of image processing literature, serves as an image-hashing method.

Perceptual Hashing can rapidly generate hash values for images by relying on the frequency characteristics of the image instead of the pixel properties. Additionally, it possesses a valuable attribute known as locality-sensitivity. This means that if images exhibit similarity, their corresponding hash values also demonstrate similarity in terms of the Hamming Distance [65]. It's important to highlight that, in the scope

Listing 4 Pseudocode of the Perceptual Hashing Algorithm [63]. A hashing method that is based on using discrete cosine transform (DCT) for comparing images based on frequencies rather than direct color values.

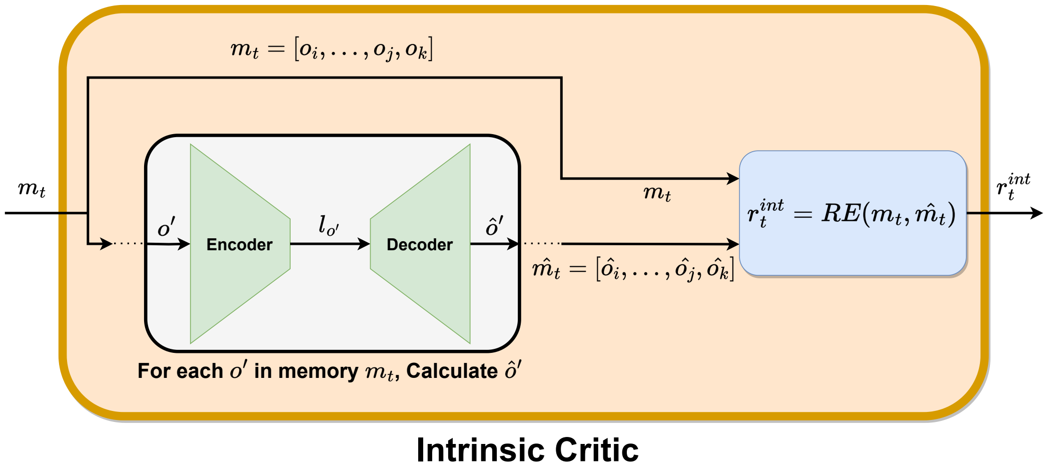
```
Algorithm Input: Image with an arbitrary size of pixels.
1. Resize the image to 32x32 pixels.
2. Convert the image to grayscale.
3. Compute the Discrete Cosine Transform (DCT) of the image.
4. Extract the top-left 8x8 pixels from the DCT (excluding the first row and column).
5. Compute the average value of these 8x8 pixels.
6. For each pixel in the 8x8 block:
      • If pixel_value > average:
          – Set bit to 1.
      • Else:
          – Set bit to 0.
7. Construct the hash based on the binary bit sequence.
Algorithm Output: 8-byte integer hash value.
```

of this work, none of these properties are presently being utilized. These aspects are introduced for potential consideration in future research endeavors. In Listing 4, the pseudo-code of the perceptual hashing algorithm is given.

However, it's important to know the potential limitations of this approach. As an approximate hashing method, Perceptual Hashing may cause collisions in hash values (with distinct observations resulting in the same hash value). Such collisions could lead to incorrect calculations and negatively affect learning. Furthermore, in noisy environments, minor changes could lead to significant differences in hash values, resulting in many different hashes, and even the locality-sensitivity property does not hold for these observations. In the experiments conducted, none of these drawbacks were observed. However, future applications should take into account these possible pitfalls as part of their considerations.

It's important to note that this study focuses on perceptual hashing due to the effective and fast hashing of images. Also, in future works, further intrinsic motivations can be designed in terms of calculating novelty by using hash similarity and mean hash distance of observations.

3.4.2 Novelty-Based Intrinsic Motivation with Autoencoder Reconstruction Error



$m_t = [o_i, \ldots, o_j, o_k]$
m_t
m_t
o'
Encoder
$l_{o'}$
Decoder
$\hat{o}'$
$\hat{m}_t = [\hat{o}_i, \ldots, \hat{o}_j, \hat{o}_k]$
$r_t^{int} = RE(m_t, \hat{m}_t)$
r_t^{int}
For each o' in memory m_t, Calculate $\hat{o}'$
Intrinsic Critic

Figure 3.4: Novelty-based Intrinsic Critic Module: Intrinsic reward calculation by using autoencoder reconstruction error of the memory elements.

To measure the novelty of memory, the inherent characteristic of an autoencoder's ability to poorly reconstruct unfamiliar data can be utilized. An autoencoder network is a specialized network for utilizing unsupervised learning where it learns to compress the input data (o) and subsequently attempts to reconstruct it, denoted as $\hat{o}$, solely based on the compressed latent representation z. (Equations 3.3a, 3.3b and 3.3c). Autoencoders typically employ neural networks to embody both the encoder (compression) and the decoder (decompression) functions. They are generally trained to reproduce the input with high fidelity, and thus, they tend to perform well on samples they have already been trained on. However, when faced with novel, unseen samples, autoencoders may struggle to reconstruct the input accurately. This property can be used as a measure of novelty detection. The effectiveness of the reconstruction is dependent on both the structure of the neural network and the optimization algorithm used. Training, in simplest terms, involves optimizing the neural network weights to minimize a Loss function $L(o, \hat{o})$, where it is often defined as the metric of the reconstruction error.

The performance, how well the autoencoder reconstructs the inputs, depends on the training itself. Given enough data and enough expressivity of the given neural networks, an autoencoder can optimize its parameters where the loss is nearly optimal. This argument holds when the data is equally distributed, but when the distribution of the data becomes uneven, autoencoders tend to generate lower reconstruction errors on examples that are sampled more frequently.

In reinforcement learning problems, states/observations are not always equally explored. Even with the intention of uniform exploration of states, the observations are sampled unevenly due to perceptual aliasing, where some states map to the same observation. In addition, there might be some partially observable cases where some information is only available at some point, which needs to be remembered.

In case of this imbalance problem, autoencoders can be used to measure the novelty of a given sample. For novel states/observations, autoencoders have higher reconstruction loss (also known as reconstruction error) since the network is trained mainly on very frequent states/observations. This reconstruction loss can be used as an intrinsic reward to increase learning performance. In [66], authors successfully used this approach and conducted detailed experiments on many environments.

It can be argued that the use of autoencoders for novelty can be extended to sequences of observations, namely a memory, that represent experiences in an environment. As memory becomes more distinctive, an autoencoder trained in the environment must generate a higher reconstruction error for the memory (Equation 3.3c). A memory-based RL agent may utilize the novelty of its memory to get informed about how distinctive its current experience is in the environment (Equation 3.3e). Then this measurement of the memory novelty can be used as an intrinsic reward to support learning how to effectively manipulate a memory (Equation 3.3f). Similar to the frequency-based intrinsic motivations, given the constantly changing reconstruction losses, the scaling of the intrinsic reward with the extrinsic reward presents a challenge without adaptive reward scaling. Instead of this direct scaling, the intrinsic reward can be dynamically normalized by using a running mean (as described in

Equation 3.3g) and variance (referenced in Equation 3.2d). This normalization scales intrinsic rewards into a zero mean and unit variance distribution. Based on this normalization (shown in 3.3i), negative normalized values indicate that the original value is below the running mean. A value of zero signifies equivalence to the average, and a positive value indicates that it exceeds the running mean. In order to positively motivate agents to retain novel observations in memory, the normalized intrinsic reward values are clipped to fall within the range of [0, 1]. The total combined reward is defined in Equation 3.3j.

$$z = Encoder(o) \tag{3.3a}$$

$$\hat{o} = Decoder(z) \tag{3.3b}$$

$$\hat{o} = Autoencoder(o) = Decoder(Encoder(o)) \tag{3.3c}$$

$$\hat{m_t} = [\hat{o_i}, ..., \hat{o_j}, \hat{o_k}] \tag{3.3d}$$

$$RE(m_t, \hat{m_t}) = \frac{1}{c} \sum_{(o', \hat{o})' \in (m_t, \hat{m_t})} \|o' - \hat{o}'\| \tag{3.3e}$$

$$R^{int}(m_t) = RE(m_t, \hat{m_t}) \tag{3.3f}$$

$$\mu_t = \lambda \mu_{t-1} + (1 - \lambda) R^{int}(m_t) \tag{3.3g}$$

$$\sigma_t^2 = \lambda \sigma_{t-1}^2 + (1 - \lambda)(R^{int}(m_t) - \mu_t)^2 \tag{3.3h}$$

$$\widetilde{R}^{int}(m_t) = clip(0, \frac{R^{int}(m_t) - \mu_t}{\sigma_t}, 1) \tag{3.3i}$$

$$\bar{R}_t = R_t + \beta \widetilde{R}^{int}(m_t) \tag{3.3j}$$

Let o represent the observation and z denote the compressed latent vector. The reconstructed observation is given by $\hat{o}$, while $\hat{m_t}$ stands for the reconstructed memory, which encompasses the reconstructed observations. The function $RE(m_t, \hat{m_t})$ is the reconstruction error function, which computes the average reconstruction error between each observation and its corresponding reconstructed observation. The term c defines the memory capacity, known as the length of the memory, $c = \|M\|$. Additionally, $\|o' - \hat{o}'\|$ represents the error norm, which can either correspond to the L1 or L2 norm. The zero-mean normalized and clipped intrinsic reward is given by

$\widetilde{R}^{int}(m_t)$, with its values ranging from $[0, 1]$. A value greater than zero for this term indicates that the intrinsic reward surpasses the average. This clipping ensures that agents are not motivated for intrinsic rewards falling below the average. The terms μ_t and σ_t denote the running average and variance values, respectively, at time t. Such that $\mu_0 = 0$ and $\sigma_0^2 =$ Lastly, $\lambda \in [0, 1]$ serves as the running average adaptation hyperparameter. Figure 3.4 illustrates the intrinsic critic module using these formulations.

It is possible that an autoencoder network can be trained simultaneously with a reinforcement learning policy through the utilization of distinct loss functions and optimizers. Experimental and algorithmic details and results are explained in Chapter 4.

3.5 LSTM Based Internal Hidden State Memory Augmentation

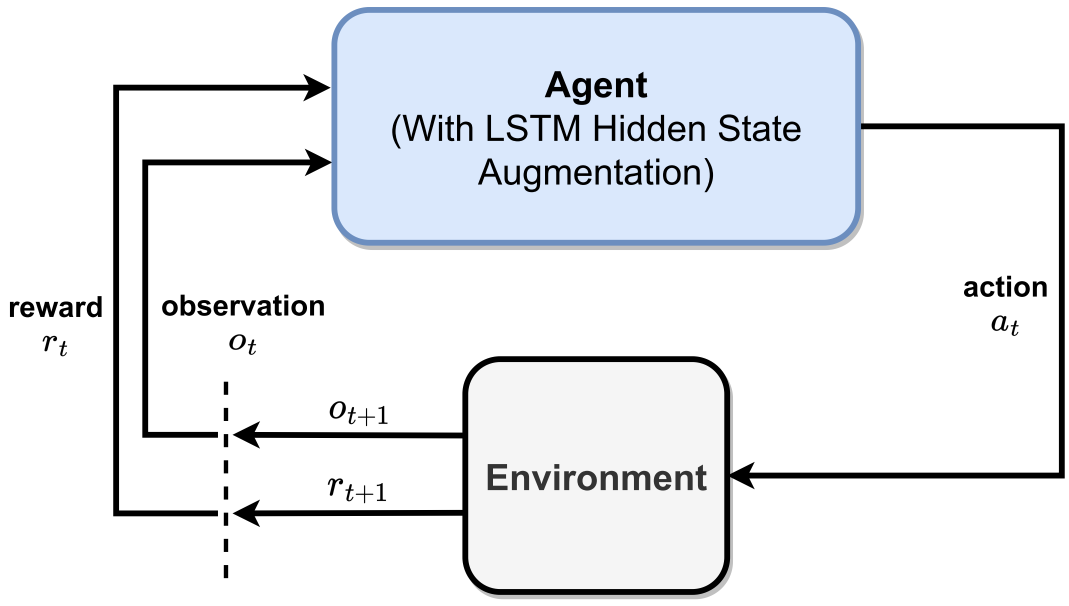


Figure 3.5: Visualization of the LSTM-based internal hidden state augmentation: A Deep RL agent is augmented with LSTM Networks for an internal memory representation of the hidden state.

As defined earlier in Section 2.4.2, the inclusion of a memory representation or context is imperative for enhancing a hidden state, facilitating the optimal solution of POMDP problems. In Section 3.2, an external memory was defined for visual reinforcement learning tasks, representing memory external to the agent. As an alternate approach, the integration of deep reinforcement learning can leverage Recurrent Neural Networks, specifically LSTM Networks, due to their capability to remember long-term information to augment an internal hidden state within the policy network of the agent.

In Section 2.5, LSTM networks are defined such that:

$$(\text{output}_t, h_t, c_t) = \text{LSTM}(o_t, h_{t-1}, c_{t-1})$$

In this formulation, h_t is the lstm hidden state (interpreted as the short-term memory/the context), and c_t is the cell state (interpreted as the long-term memory). As previously mentioned, hidden and cell states are dependent on each other and updated in each LSTM update. Lastly, h_t can be further used to calculate the output_t value. By using the lstm hidden state h_t of the LSTM network, the policy $\pi_\theta(a_t|h_t)$ and/or the value $V_\theta(h_t)$ function networks (depending on the approach/algorithm) augment a hidden state MDP.

The LSTM, along with the policy and/or value function networks, can be trained using one of two methods. The first involves separate neural networks where the hidden state h_t is utilized as an input for the latter. Alternatively, these elements can be directly trained within a single, unified neural network. An illustration of the LSTM-based hidden state augmented agent is shared in Figure 3.5.

Choosing between internal and external memory representations often comes down to the specific requirements and constraints of the given task. For simpler tasks with shorter-term dependencies, an internal memory representation might be sufficient and more efficient. However, for more complex tasks requiring substantial memory capacity and the understanding of long-term dependencies, an external memory representation may be necessary. The internal memory representation is defined, implemented, and employed by utilizing LSTM networks to establish a baseline within our experiments for the proposed approaches. Details are given in Chapter 4 and later.

CHAPTER 4

EXPERIMENTAL RESULTS

In many real-world scenarios, the ability to remember and utilize distant past observations becomes paramount to achieving optimal or near-optimal policies. Such scenarios may include tasks where short-term observations are noisy, ambiguous, or provide only incremental information about the environment. In others, the agent may need to recall specific past events to make informed decisions in the present, especially when the immediate observations are insufficient or misleading. In this chapter, the selected/designed POMDP tasks are defined within simulation environments. Reinforcement Learning algorithms are formulated to solve these tasks, and experimental results are subsequently shared.

4.1 Environment Design and Selection

In general, choosing an environment that embodies these characteristics is crucial when evaluating the efficiency of algorithms and techniques designed for POMDPs. Aspects to consider when designing or selecting such POMDP environments include:

- **Observation Ambiguity:** The environment should frequently present observations that are ambiguous without the context of past information, forcing the agent to rely on its memory to disambiguate them.
- **Reward Sparsity:** Infrequent rewards will ensure that the agent has to leverage its memory to make sense of its actions and the outcomes, particularly when the rewards are contingent on sequences of actions or events spread out over time.
- **Noise and Distractions:** Introducing stochasticity in observations or adding

irrelevant details can test an agent's capability to filter out noise and focus on pertinent information.

- **Extended Temporal Dependencies:** The environment should be structured such that actions taken at one point might have consequences much later. Possibly, the past knowledge is needed in a temporally distance place in the sequence, compelling the agent to maintain and access a long-term memory of events.

In this study, only a special subset of POMDP problems is focused on, which requires long-term remembering of visual observations to obtain optimal policies. In the following, several problems formulated within this context:

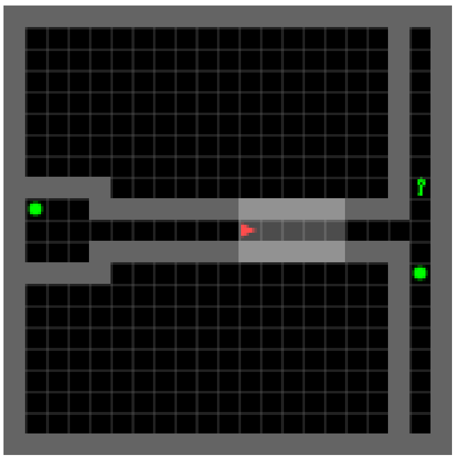


Figure 4.1: An example illustration of the fully observed `Memory Environment`. The agent is marked with a red triangle and the partial observation of the agent is represented by the light shaded area.

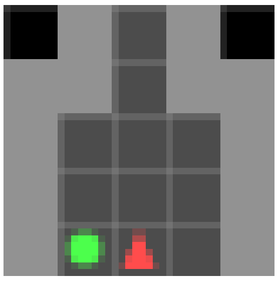


Figure 4.2: An example observation in `Memory Environment` where the agent is in the initial state. The object in green (either a ball or a key figure, randomly initialized at each episode) marks the hint object, which the agent needs to remember until the end of the maze to optimally solve the environment.

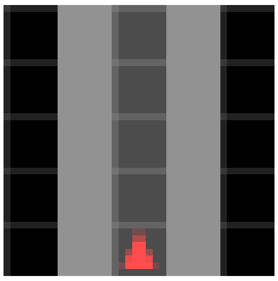


Figure 4.3: An example observation in `Memory Environment` where the agent is in the corridor. Due to the visual occlusions from partial observability, many states are mapped to this same observation.

- **`Memory Environment`** ([67]):

  This environment can be defined as the visual implementation of the T-Maze Environment [61]. The agent begins in a small room with an object, then navigates a narrow hallway that leads to a split with an object at each end, namely a t-shaped junction. In this intersection, the agent needs to select one of the directions, and the hint of this correct choice is only given at the beginning of the episode. The object which makes the agent successful is randomly selected at each episode, and the hint object changes accordingly. Therefore, in an optimal policy, an agent needs to observe and remember this single information until the end. For the purpose of explanation, the fully observable setting of this environment is visualized in Figure 4.1. In the partially observable setting, the agent can only see nearby objects, so the hint object cannot be seen at all times,

which the problem becomes a POMDP. In addition to POMDP, high dimensional inputs, sparse rewards, and visual occlusions make this simple problem domain very hard to solve. Hence, it is believed that this domain serves as a strong baseline environment for evaluating algorithms designed for POMDP problems, particularly with regard to sample efficiency. Figures 4.2 and 4.3 shows example partial observation images taken from this environment.
In more detail, the environment is defined as follows:

- The observation space is 3 dimensional visual RGB image $(N, N, 3)$ where N is the pixel size.

- The action space consists of 3 possible actions, which are "Turn Left, Turn Right, Move Forward".

- An episode ends if any following conditions are met:
  * The agent reaches either the correct or wrong matching object at the end of the maze.
  * The total timesteps exceed the environment timeout value.
- Lastly, the reward is defined in Equation 4.1

$$R = \begin{cases} 1 - 0.9(\text{step\_count}/\text{max\_steps}), & \text{if reached correct matching object} \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

• **`Mortar Mayhem Grid Environment`** ([68]):
In Memory Environment, by the problem definition, the agent needs to remember a single observation to solve this task optimally. However, in order to show the scalability in terms of memory size greater than one, a more complex environment, namely `Mortar Mayhem Grid`, is also selected and defined. Mortar Mayhem Grid is a visual environment where the agent lives in a square-shaped grid. At the start of an episode, a sequence of commands appears in the visual observation, which are visual direction signs that denote how the agent should move. At the start of an episode, these commands are visualized one by

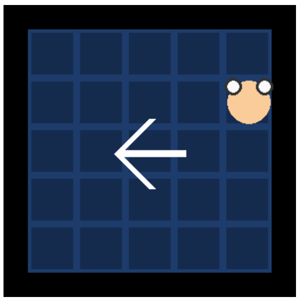
one and never appear after that. The agent should remember and execute these sequences of commands in the right order to optimally solve this problem. Similar to the Memory Environment, high dimensional inputs, sparse rewards, and visual occlusions make this simple problem domain very hard to solve, and the requirement of remembering an ordered sequence of observations is crucial for experimentations and comparisons. Figures 4.4 and 4.5 shows example partial observation images taken from this environment.
In more detail, the environment is defined as follows:

- The observation space is 3 dimensional visual RGB image $(N, N, 3)$ where N is the pixel size.
- The action space consists of 4 possible discrete actions, which are "No Operation, Turn Left, Turn Right, Move Forward".
- An episode ends if any following conditions are met:
  - $*$ The agent correctly executes all commands in a given time period.
  - $*$ The agent falsely executes a command at any time step.
  - $*$ The total timesteps exceed the environment timeout value.
- Lastly, the reward is defined in Equation 4.2

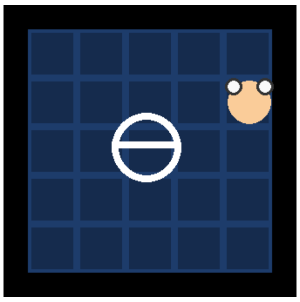
$$R = \begin{cases} R^C_{\text{success}} * \hat{C}, & \text{if successfully executed first } \hat{C} \text{ commands,} \\ R^C_{\text{success}} * C + R^E_{\text{success}}, & \text{if successfully executed all commands,} \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

  Where:

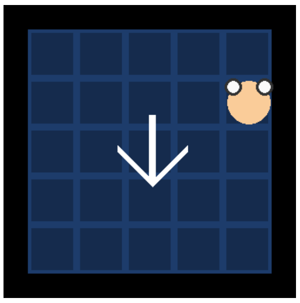
  - $*$ C is the total number of commands defined in the environment.
  - $*$ $0 \leq \hat{C} < C$ is the number of successfully executed commands, which is lower than the total number of commands.
  - $*$ R^C_{success} is the reward value when the execution of a command is a success,
  - $*$ R^E_{success} is the reward value when all commands are successfully executed.



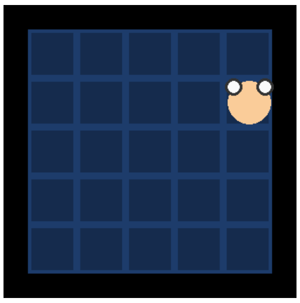
(a) 1st Observation: The command is given at the center of the visual observation (Move Left)



(b) 2nd Observation: The command is given at the center of the visual observation (No Operation)



(c) 3rd Observation: The command is given at the center of the visual observation (Move Down)



(d) 4th observation: The sequence of commands is all given, and now the agent needs to execute the sequence of actions accordingly.

Figure 4.4: Example initial observations in Mortar Mayhem Grid Environment. The agent needs to remember given commands which are illustrated as signs in the center of the visual observation. This example is given for an environment where the number of commands $C = 3$.

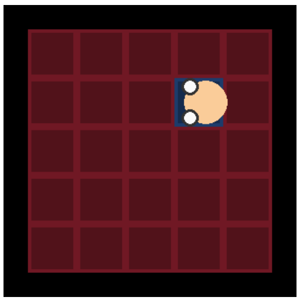


Figure 4.5: An example observation in Mortar Mayhem Grid Environment where the agent executes a correct command and all of the tiles is visualized in red except the agent is standing on if the agent is in the right position. At this step, the agent gains a reward and continues to remember/execute the correct commands.

In summary, these environments were chosen to provide a fair comparison between the proposed methods and the state-of-the-art. Further in this chapter, algorithms and test scenarios are outlined for comparative analysis.

4.2 Algorithms

After discussing the environment design in previous sections, this section delves into the decision-making process of the agents. In this study, we opted for one of the most widely used RL algorithms, specifically the Proximal Policy Optimization (PPO) algorithm. This decision was guided by the intention to emphasize the memory architectures rather than the reinforcement learning algorithm itself.

Nevertheless, the proposed architectures are versatile and can be integrated with any deep reinforcement learning algorithm. It's important to highlight that these architectures aren't restricted solely to the PPO algorithm. The experiments are based on the PPO algorithm with convolutional and fully connected neural network layers, similar to the study in [34]. However, given the potential challenges due to the partial observability of the environment designs, direct utilization of the vanilla PPO algorithm is not sufficient.

As previously defined in Sections 3.3, 3.4, and 3.5, the internal and external memory mechanisms are extended to the practical implementation of the PPO algorithm. This extension is aimed at resolving the POMDP environments at hand.

Each of these architectural extensions aims to harness both the agent's past experiences and intrinsic motivations, ensuring that decision-making is based on immediate observations and informed by historical data and internal drives. In the following section, practical methods to solve the defined environments are outlined, and these methods are detailed extensively:

- **PPO + VSMM:**
  This method augments the existing PPO algorithm with Visual Self Memory Management (Section 3.3) to create an external memory for agents in reinforcement learning problems. Augmentation of the external memory is defined by extending the observation space with current observation and external memory observations. As previously defined, agents also have the ability to manipulate the memory contents. Therefore, the action space is also extended. From the environment perspective, the agent is still compatible since memory observations and actions are only related to the agent itself. Therefore this algorithm can be used in any reinforcement learning environment.

  One main drawback of this algorithm is the increasing number of observation and action dimensions. Due to the curse of dimensionality, observation/action space volumes grow exponentially, and the required number of samples/data drastically increases.

  Even though the environment-related observation/action spaces do not change, memory-related changes negatively affect the overall reinforcement learning problem. Consequently, "PPO + VSMM" is further enhanced through the integration of two different intrinsic motivation strategies, intended to eliminate the negative effects mentioned earlier.

- **PPO + VSMM With Frequency Based IM:**
  A count/frequency-based intrinsic motivation can be helpful with VSMM to motivate agents to keep novel observations in their memory. It's worth mentioning that the primary purpose of this approach is not to improve the overall exploration. Instead, it only motivates agents to use memory for their benefit. For

visual problems where the observation space is discrete but high-dimensional, effective calculation of the observation counts is required to make the algorithm be tractable. This challenge is surpassed by employing a perceptual hashing algorithm to efficiently hash and store observation counts. Further details can be found in Section 3.4.1.

- **PPO + VSMM With Autoencoder Based IM:**
  As an alternative approach, the novelty of external memory can be measured by utilizing the inherent characteristics of an autoencoder's ability to reconstruct unfamiliar data poorly. Similar to frequency-based IM, this metric can be further used to motivate agents to utilize external memories properly. In addition to the frequency-based IM methods, since neural network-based autoencoders are used for calculating novelty, there is no observation limitation, which can be useful in large dimensional and continuous problems. Details are given in Section 3.4.2.

- **PPO + LSTM:**
  Instead of utilizing an external memory, recurrent neural networks can also be adapted with the PPO algorithm to create an internal memory. For this problem, one of the state-of-the-art methods for remembering long-term information is utilized, specifically the LSTM Networks. Details are given in Section 3.5.

For simplicity, the naming of these approaches is restricted solely to their additional mechanisms. For instance, rather than using the name "PPO + LSTM", the method is simply referred to as "LSTM" in the reinforcement learning experiments. This is because all the algorithms utilize PPO, making the inclusion of the "PPO" term in algorithm names redundant.

4.3 Environment Setup

- Simulation Environments and VSMM Details:
  - For both simulation environments (Memory and MMG environments), normalized visual observations with range [-1, 1] and pixel shape of (84, 84, 3) have been used.

  - The external memory has been formalized as a 4-dimensional array denoted as (memory_capacity, 84, 84, 3), wherein memory capacity assumes a positive integer value.
  - Observation space with external memory is formulated as a tuple (current observation, external memory).
  - In the overall neural network architecture, convolutional layers are utilized for visual observation inputs to capture spatial relationships among visual elements efficiently. After incorporating external memory, and since multiple visual images are present within a single observation, these images are processed separately.
  - Memory actions are formulated as two discrete actions, which are "No operation" and "Add to memory". The new action space is the cross-product of the existing and memory action spaces.
  - When the memory is full of observations, the replacement strategy is formulated as a FIFO (First In, First Out) Queue.
  - For training stability, intrinsic motivation rewards start after 200.000 timesteps for all environments. Until this timestep, frequency-based or autoencoder-based intrinsic motivation functions start bootstrapping. These functions always calculate IM reward values but are not applied to the agent until this bootstrapping phase is over.
  - In Autoencoder Based Intrinsic Motivation approaches, the autoencoder network is concurrently trained with the reinforcement learning policy by using separate loss functions and optimizers.

- Experimental Conditions:
  - The seeds for all experiments are initialized randomly to ensure uniformity across multiple runs. For Memory Environments, 16 parallel runs are carried out, and for MMG (Memory Management Games) Environments, 12 parallel runs are executed, each with randomly initialized seeds.
  - The same hyperparameters and network architectures are applied to all algorithms and experiments. In the context of ablation study, only the architectural extensions are altered, while the shared hyperparameters and

neural network architectures remain consistent across all algorithms. This approach enables a relative performance comparison between the algorithms.

- Baseline and Comparisons:
  - Memory Environment: Due to the partial observability, the vanilla PPO algorithm (classical PPO algorithm without any recurrence or memory) can only learn to navigate in the maze, but it cannot correctly decide the t-junction direction since there is no available information at decision step. This results in the decision at the t-junction becoming random and converging to an average reward of 0.5 in the Memory Environment. Anticipatedly, all the formulated algorithms are projected to surpass this baseline and converge towards an optimal policy, approaching a reward value of 1.
  - MMG Environment: Due to the partial observability, the vanilla PPO algorithm returns the worst possible reward, a value of "-1" on average in this environment. Since this environment requires remembering multiple observations in order to obtain the optimal policy, an agent needs to put all of the required observations into their memory and act accordingly. It is anticipated that all the devised algorithms will outperform the vanilla algorithm and push the environment reward value limits.
  - Performance Comparison: In the ablation study, since all of the common hyperparameters and neural network architectures are the same, algorithm performances in terms of sample efficiency can be relatively compared. The term sample efficiency simply means that how many environment steps are required on average to converge to the optimal policy.
  - Generalization Comparison: Throughout the training process, periodic evaluations of algorithm performance are conducted, creating metrics such as mean rewards, episode lengths, and others. Furthermore, within each evaluation interval, the algorithm's policy is also evaluated across a different environment configuration. For the memory environment, this generalization comparison is performed by utilizing a different environment configuration, where the maze length is 10 more than the current envi-

ronment configuration, shared in Figure 4.6. This allows us to see the generalization capability of the algorithms for unseen environment configurations.

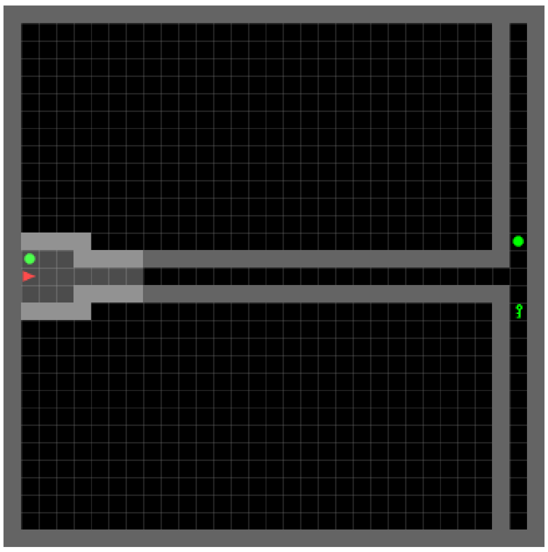


Figure 4.6: Generalization Comparison: An example of a different environment configuration where the maze length is 10 more than the current environment configuration, as previously shared in Figure 4.1.

Python 3.11 [69], the most current version available at the time, is used for all software implementations and experiments. In addition, an open-source reinforcement learning framework named "Ray RLlib" [70] has been used to utilize multiple GPUs and CPUs efficiently. In the context of the deep learning framework, the PyTorch [71] tensor library was used for efficient computational operations.

4.4 Results and Discussion

In this section, various experiments are performed to assess and evaluate the performance of given memory architectures. In summary, the following sections are structured as follows:

- Ablation study of comparing external and internal memory architectures. (Memory and MMG Environment)

- Memory Environment: Comparison of memory architectures in terms of generalization capability for different environment configurations.
- Memory Environment: Analyzing the effects of having an excess number of external memory size.

4.4.1 Ablation study of comparing external and internal memory architectures

During the ablation study, the anticipated outcomes involve observing the performance impacts of different memory architectures. These results are achieved solely by altering the architectural extensions while retaining the same reinforcement learning algorithm (PPO), base neural network structure (including convolutional layers and MLP), and hyperparameters. Due to computational costs, total training steps are limited to 10M timesteps in the memory environment, meaning that each algorithm interacts with 10M timesteps in the given environment. Meanwhile, learning iteration occurs in batches (for example, in every 1000 timestep).

In Figure 4.8, approaches such as "VSMM, VSMM With Frequency Based IM, VSMM With Autoencoder Based IM, and LSTM" are compared in terms of average cumulative rewards per episode. This metric is calculated by averaging all the reward values achieved per episode since the previous training iteration. These algorithms aim to optimize the mean reward, which is a direct metric for the agent's performance during the training. As can be seen, all methods converge into the optimal policy, which shows that an agent can successfully use some type of memory to solve this POMDP problem optimally. In Figure 4.7, how a successful agent with an external memory operates in the Memory Environment is illustrated. Initially, the agent starts with an empty external memory. For this example, optimal behavior entails the agent adding the crucial observation to the external memory and retaining it until the t-junction step at the episode's conclusion.

As a result, VSMM alone performs the least sample efficient. It eventually converges, but it takes longer to learn the optimal behavior due to the larger observation and action spaces, making it harder to find the optimal policy. On the contrary, the contents of the external memory can always be seen and easily explained to others, which is beneficial for the safety and explainability of reinforcement learning agents. It can also be seen that it even improves overall learning stability when compared with the internal memory approach (LSTM).

On the other hand, internal memory (LSTM) based architecture performs the best in sample efficiency. However, due to the nature of neural networks, there is no clear explanation of which information is stored currently and how it is processed (it can be seen as a black-box model). In addition, in LSTM-based architecture, there is more variance among parallel runs when compared with VSMM-based approaches, which could be a problem in terms of stability.

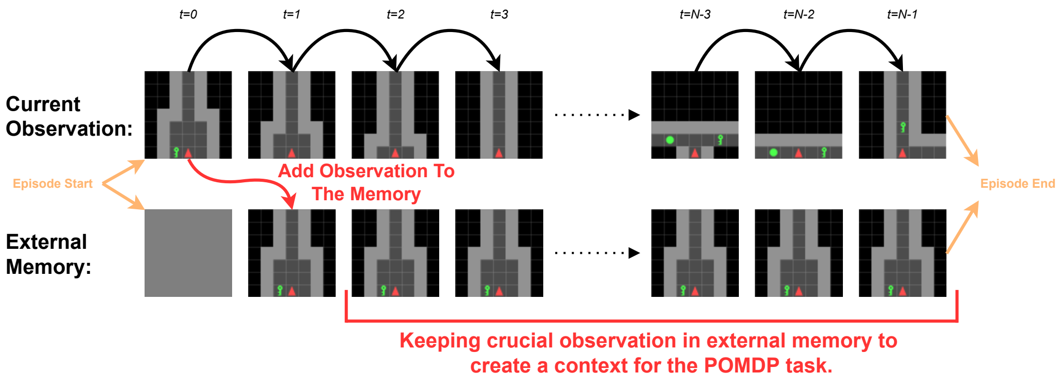

t=0
t=1
t=2
t=3
t=N-3
t=N-2
t=N-1
Current Observation:
Episode Start
Add Observation To The Memory
Episode End
External Memory:
Keeping crucial observation in external memory to create a context for the POMDP task.

Figure 4.7: Example episode in the Memory Environment with an external memory augmented agent. This figure demonstrates the current and the external memory contents of an agent for a single episode. At the start, the agent initializes with an empty external memory. In this example, the agent adds the crucial observation to the external memory and it holds until the t-junction step at the end.

The external memory architectures combined with intrinsic motivations, namely "VSMM With Frequency Based IM" and "VSMM With Autoencoder Based IM", seem to improve the disadvantages of the VSMM algorithm caused by the large observation and action space dimensions. Both approaches improve the average sample efficiency by utilizing intrinsic motivations to motivate using the external memory.

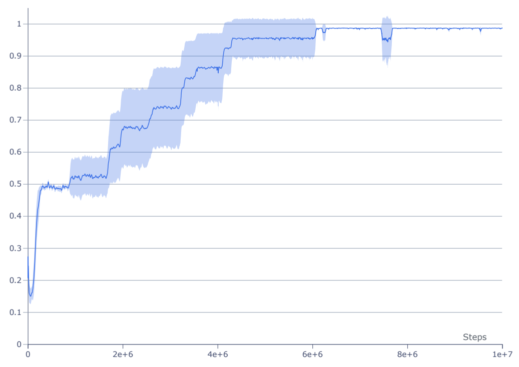
The comparison of the average episode lengths is given in Figure 4.9. This metric is calculated by averaging episode lengths since the previous training iteration. This metric itself is an indicator for showing the agent's capability of how fast it learns to navigate in the maze, but it doesn't show how optimal the agent performs in the environment since the optimal behavior depends on the agent's memory context creation capability. As can be seen, for all algorithms, all agents quickly learn to navigate through the maze to go to the t-junction, while observations are partially observable.

In Figure 4.10, for VSMM-based algorithms, the comparison of the average lengths of unchanged memory sequences per episode is given. This metric is calculated by averaging unchanged memory sequence lengths of agents with external memory since the previous training iteration. It is a good indicator of how long an agent keeps an observation through the episode. For agents utilizing external memory, the expectation is that they will remember observations for a sufficient amount of timesteps. As can be seen, all VSMM-based approaches learn to use the external memory, and more importantly, they learn to hold the important information until the t-junction. Parallel to these results, the average number of memory updates per episode is compared in Figure 4.11. The calculation of this metric is similar to previous metrics. It is expected that agents change the memory contents a few times since they need to retain and recall until the appropriate state.

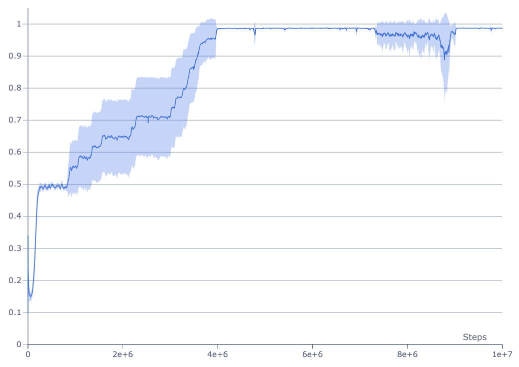
Figure and Metric Name	**Metric Value Range**	**How to evaluate?**
Figure 4.8 - Average Cumulative Rewards Per Episode	$y \in [0, 1]$	Higher value and lower variance is better.
Figure 4.9 - Average Episode Length Per Episode	$y \in [0, 1200]$	Lower value and lower variance is better.
Figure 4.10 - Average Length Of Unchanged Memory Sequences Per Episode	$y \in [0, 15]$	Higher value and lower variance is better.
Figure 4.11 - Average Number Of Memory Updates Per Episode	$y \in [0, 600]$	Lower value and lower variance is better.

Table 4.1: Summary of how to evaluate metrics for Memory Environment ablation studies.

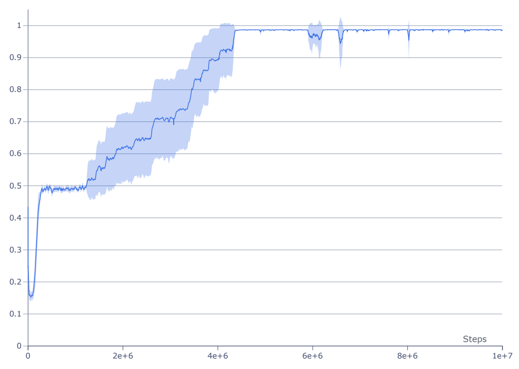
Table 4.1 serves as a summarized view of the conducted metrics, organized in a tabular format to facilitate readers' comprehension. Each figure and metric name are cross-referenced with their corresponding value ranges, along with instructions on how to interpret and evaluate the results.



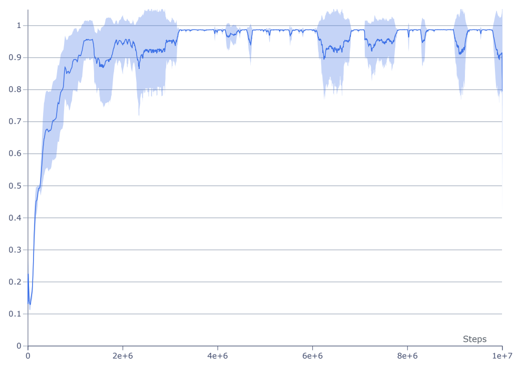
(a) VSMM



(b) VSMM With Frequency Based IM

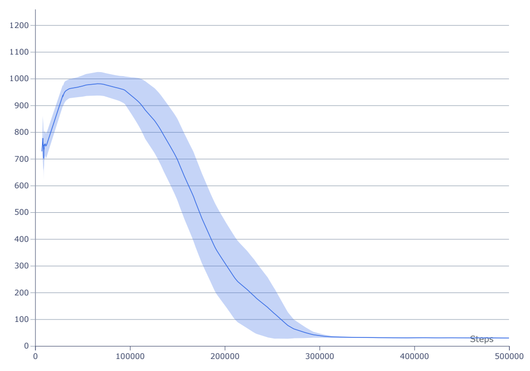


(c) VSMM With Autoencoder Based IM

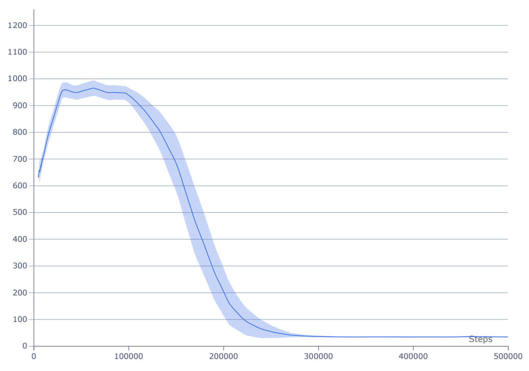


(d) LSTM

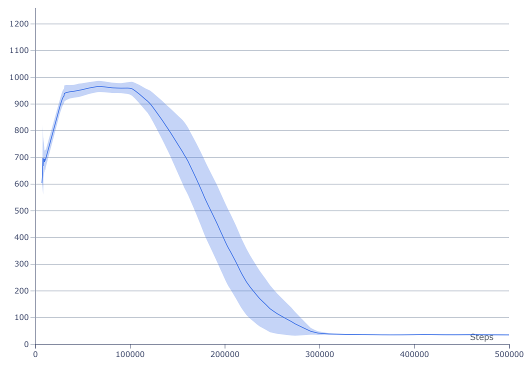
Figure 4.8: Memory Environment: Comparison of average cumulative rewards per episode. Each line represents the average of 16 parallel runs with confidence intervals.



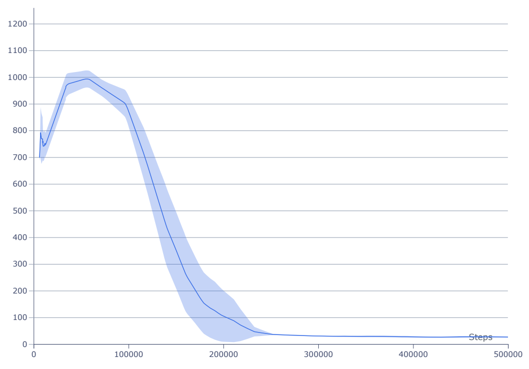
(a) VSMM



(b) VSMM With Frequency Based IM

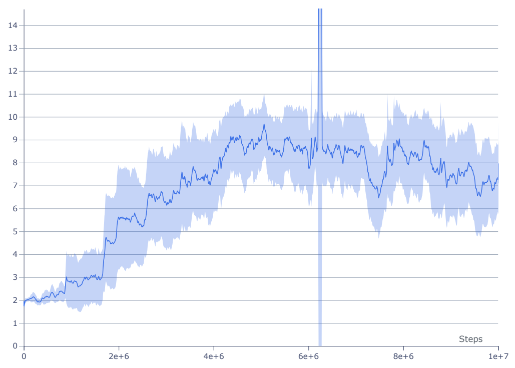


(c) VSMM With Autoencoder Based IM

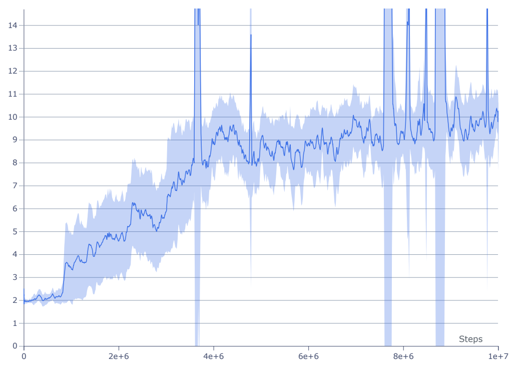


(d) LSTM

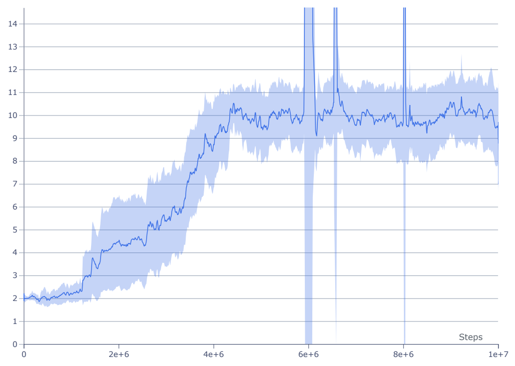
Figure 4.9: Memory Environment: Comparison of the average episode length per episode. Each line represents the average of 16 parallel runs with confidence intervals.



(a) VSMM

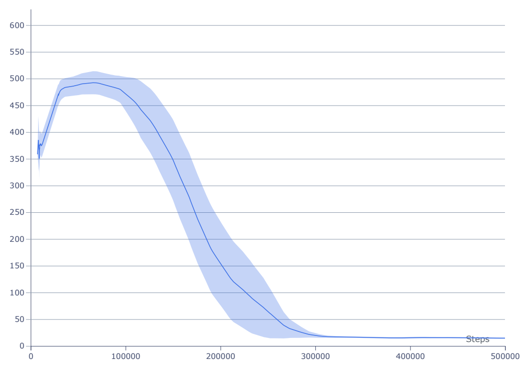


(b) VSMM With Frequency Based IM

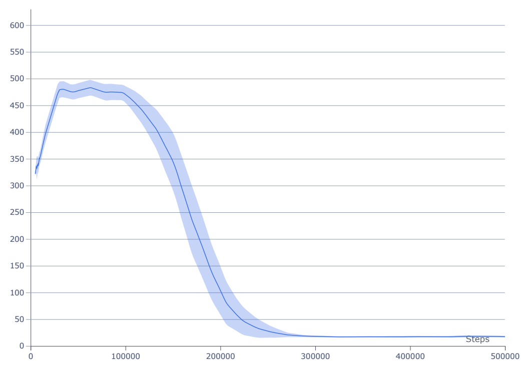


(c) VSMM With Autoencoder Based IM

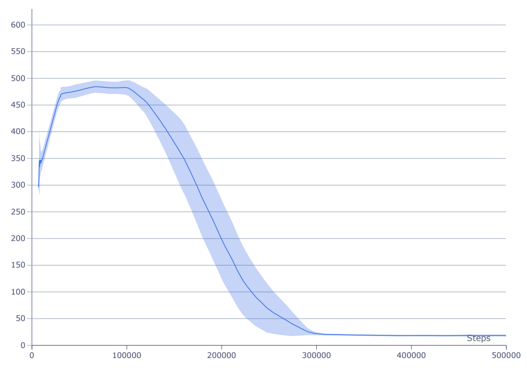
Figure 4.10: Memory Environment: Comparison of the average length of unchanged memory sequences per episode. Each line represents the average of 16 parallel runs with confidence intervals.



(a) VSMM



(b) VSMM With Frequency Based IM



(c) VSMM With Autoencoder Based IM

Figure 4.11: Memory Environment: Comparison of the average number of memory updates per episode. Each line represents the average of 16 parallel runs with confidence intervals.

To conclude the results and their discussions, by augmenting agents with external memory, agents can learn to interact with the memory mechanism, and further training allows agents to utilize the memory for solving the Memory environment visual problem. Appendix A shares detailed experiment results.

The performance effects of different memory architectures in the MMG environment are also expected to be observed by only changing the architectural extensions while keeping the same reinforcement learning algorithm (PPO) and hyperparameters. The natural structure of the MMG environment requires agents to remember multiple observations and the exact sequence throughout the episode. From the agent's perspective, the agent needs to remember correct information and order. This complexity makes the MMG environment very hard and sample inefficient for algorithms to solve. Due to computational costs, total training timesteps are limited, and the relative performance of different memory architectures with the same hyperparameters and network structure is compared.

Figure and Metric Name	**Metric Value Range**	**How to evaluate?**
Figure 4.12 - Average Cumulative Rewards Per Episode	$y \in [-1, 0]$	Higher value and lower variance is better.
Figure 4.13 - Average Episode Length Per Episode	$y \in [0, 20]$	Lower value and lower variance is better.
Figure 4.14 - Average Length Of Unchanged Memory Sequences Per Episode	$y \in [0, 15]$	Higher value and lower variance is better.
Figure 4.15 - Average Number Of Memory Updates Per Episode	$y \in [0, 15]$	Lower value and lower variance is better.

Table 4.2: Summary of how to evaluate metrics for MMG Environment ablation studies.

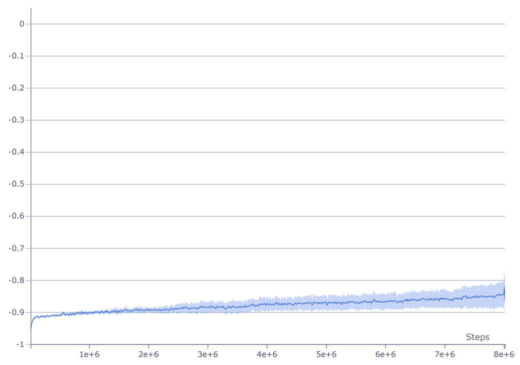
In Figure 4.12, approaches like "VSMM, VSMM With Frequency Based IM, VSMM With Autoencoder Based IM, and LSTM" are compared in terms of average cumulative rewards per episode.

As a result, VSMM alone performs the least successful approach regarding sample efficiency. Compared with other methods, VSMM could not improve its reward as much as different approaches.

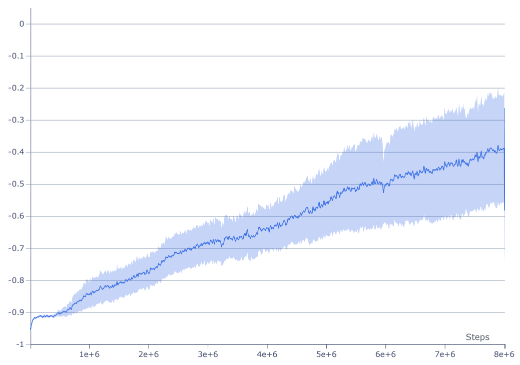
Parallel to the Memory environment results, internal memory (LSTM) based architecture performs the best in sample efficiency for this environment. Also, intrinsic motivations improve the disadvantages of the VSMM algorithm caused by the large observation and action space dimensions. Both approaches improve the average sample efficiency by utilizing intrinsic motivations to motivate using the external memory.

The average episode lengths are compared in Figure 4.13. As can be seen, results are in parallel with Memory environment results. In Figure 4.14, for VSMM-based algorithms, the comparison of the average lengths of unchanged memory sequences per episode is given. As can be seen, all VSMM-based approaches learn to use the external memory, and more importantly, it learns to hold the important information until the t-junction. Parallel to these results, the average number of memory updates per episode is compared in Figure 4.15.

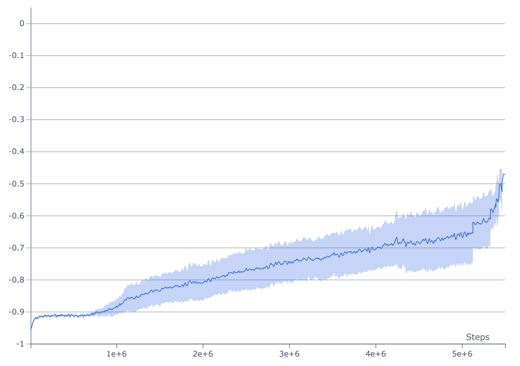
In table 4.2, the defined metrics are summarized in a table format to ease the understanding for readers. Each figure and metric name is referenced with corresponding value ranges, and the method of evaluating the results is denoted.



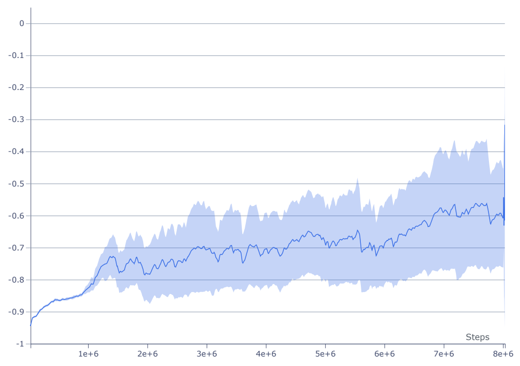
(a) VSMM



(b) VSMM With Frequency Based IM

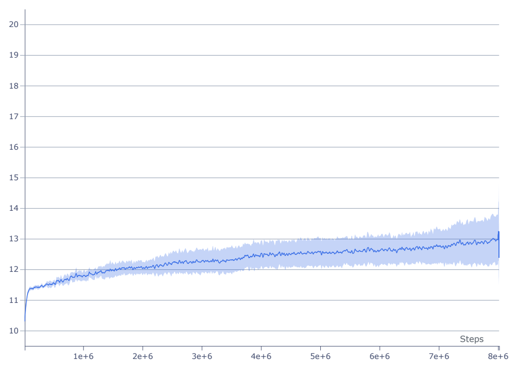


(c) VSMM With Autoencoder Based IM

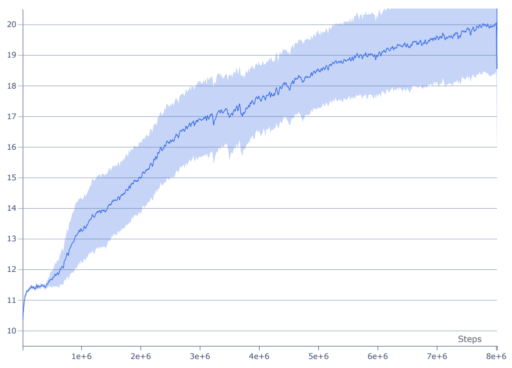


(d) LSTM

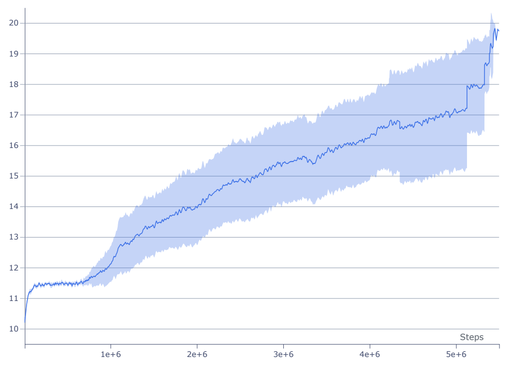
Figure 4.12: MMG Environment: Comparison of average cumulative rewards per episode. Each line represents the average of 16 parallel runs with confidence intervals.



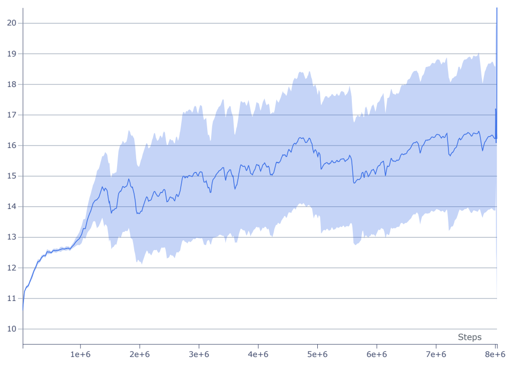
(a) VSMM



(b) VSMM With Frequency Based IM

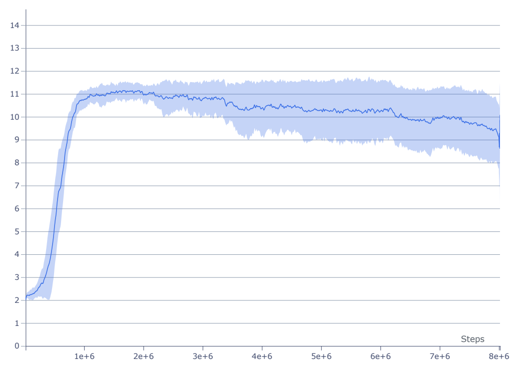


(c) VSMM With Autoencoder Based IM

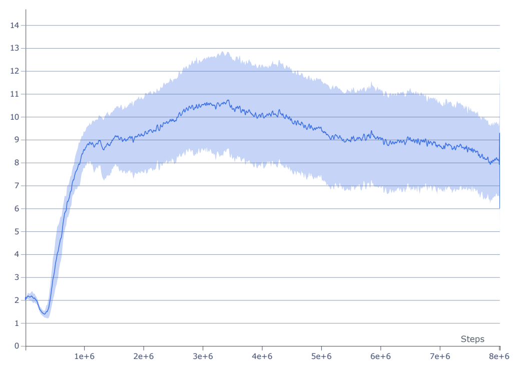


(d) LSTM

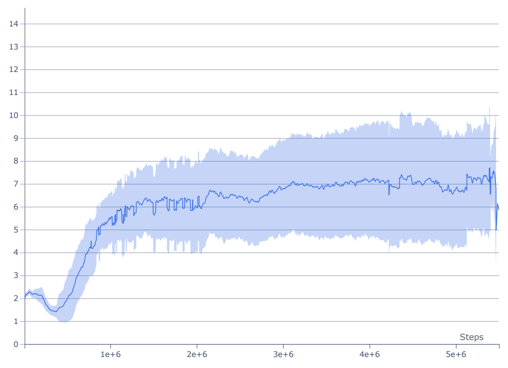
Figure 4.13: MMG Environment: Comparison of the average episode length per episode. Each line represents the average of 16 parallel runs with confidence intervals.



(a) VSMM

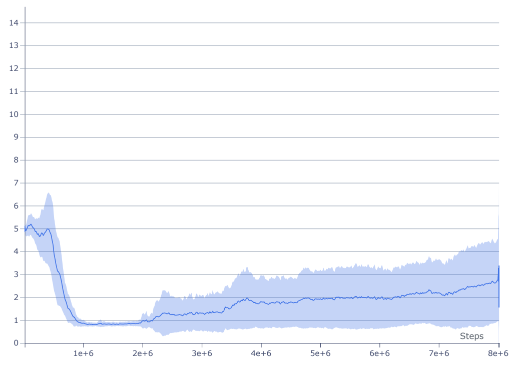


(b) VSMM With Frequency Based IM

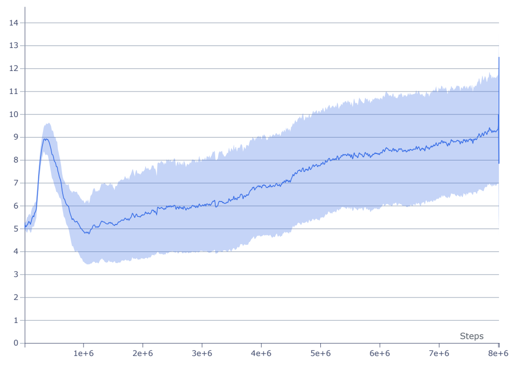


(c) VSMM With Autoencoder Based IM

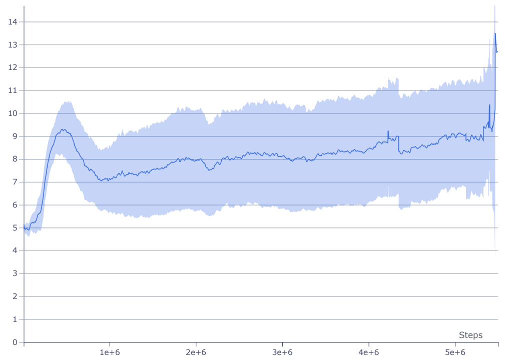
Figure 4.14: MMG Environment: Comparison of the average length of unchanged memory sequences per episode. Each line represents the average of 16 parallel runs with confidence intervals.



(a) VSMM



(b) VSMM With Frequency Based IM



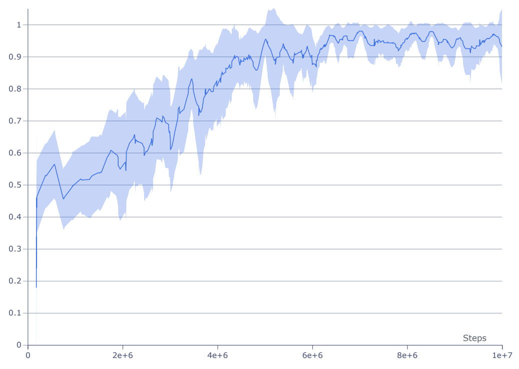
(c) VSMM With Autoencoder Based IM

Figure 4.15: MMG Environment: Comparison of the average number of memory updates per episode. Each line represents the average of 16 parallel runs with confidence intervals.

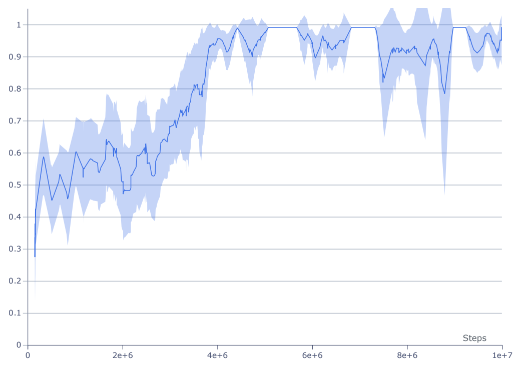
To conclude, MMG environment results are very similar to the Memory environment. Due to the computational limitations, the earlier iterations of the algorithms are relatively compared, and even in this situation, the results are coherent. Lastly, additional experiment results are shared in Appendix A.

4.4.2 Memory Environment - Comparison of memory architectures in terms of generalization capability for different environment configurations

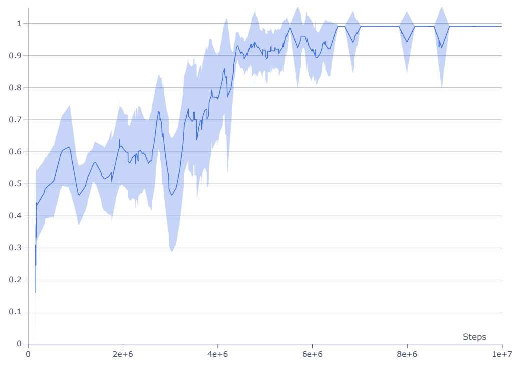
In this section, the generalization capabilities of the algorithms are further analyzed and compared with the internal memory-based approach. Evaluation for additional scenarios using the same comparison metrics mentioned in Section 4.4.1 is conducted.



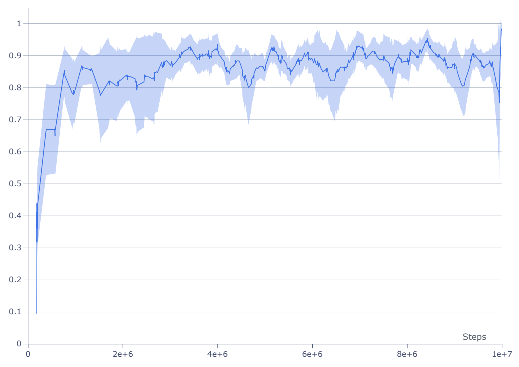
(a) VSMM



(b) VSMM With Frequency Based IM



(c) VSMM With Autoencoder Based IM



(d) LSTM

Figure 4.16: Memory Environment - Generalization Scenario: Comparison of average cumulative rewards per episode. Each line represents the average of 16 parallel runs with confidence intervals.

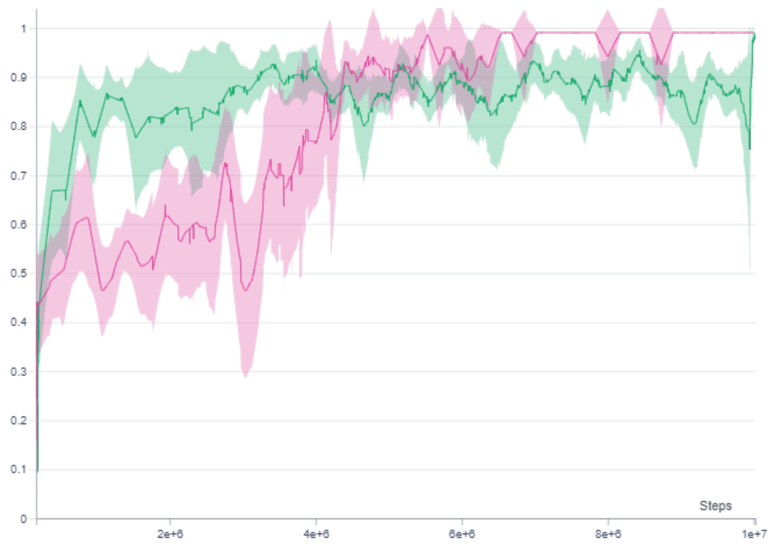


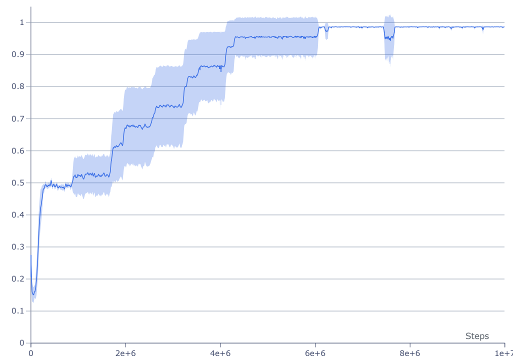
Figure 4.17: Memory Environment - Generalization Scenario: Comparison of average cumulative rewards per episode, best two approaches are depicted in the same figure. Each line represents the average of 16 parallel runs with confidence intervals. Green is Internal Memory Based Approach (LSTM) and Pink is VSMM With Autoencoder-Based Intrinsic Motivation.

As previously defined, algorithm performances are periodically evaluated in terms of given metrics during the training. In this section, a new comparison scenario for the generalization capabilities of algorithms is introduced. The algorithm policy is also evaluated for a different environment configuration in each evaluation period, where the maze length is 10 more than the current environment configuration. This approach demonstrates the generalization capability of the algorithms for unseen environment configurations. In other words, the agent is not trained in any generalization environments; this configuration is solely used for evaluation.

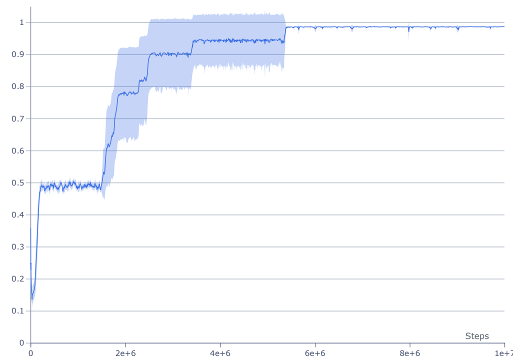
It is believed that one of the most important results of the study is this experiment. In Figure 4.16, the LSTM-based internal memory approach is rigorously compared with the VSMM-based external memory strategy. Immediately, a sharp contrast emerges: the LSTM-based methodology, despite its prevalence in many applications, falls notably short in achieving the same average rewards as the VSMM-based approach. This isn't a minor discrepancy but a substantial one. The inherent strength of the

VSMM-based method stems from its powerful generalization capabilities. By leveraging an external memory structure, it can adeptly capture and retain rare observations, which can be pivotal in decision-making scenarios. In contrast, LSTM's internal memory might not capture these nuances as effectively, and it seems learned policy is memorized to create a context for certain timesteps only. This observation underscores the potential superiority of external memory systems, especially in environments where nuances and rare observations can drastically influence outcomes in transferring the learned policy into new configurations. To delve deeper into the empirical evidence and comparison, readers are encouraged to consult Appendix A.

4.4.3 Memory Environment - Analyzing the effects of having an excess number of external memory size



(a) VSMM ($\|M\| = 1$)



(b) VSMM ($\|M\| = 3$)

Figure 4.18: Memory Environment: Comparison of average cumulative rewards for different memory sizes ($\|M\| = 1$ and $\|M\| = 3$). Each line represents the average of 16 parallel runs with confidence intervals.

As previously mentioned, agents are required to remember a single observation in the memory environment to achieve optimal behavior. This section examines the effects of an excess external memory size. For this experiment, the SMM approach with external memory sizes of $c = 1$ and $c = 3$ is compared. In Figure 4.18, the comparison of average cumulative rewards per episode metric is illustrated. As depicted, the SMM with a larger memory (memory size of 3) outperforms the classic SMM configuration (memory size of 1) in terms of sample efficiency.

From the extended experiments, it is observed that, with a longer memory, agents can exhibit greater robustness to mistakes when deciding on memory actions. This is because the observation remains in the memory for an extended period due to the FIFO (first in, first out) replacement strategy, as mentioned in Section 4.3. Thus, based on these findings, it is inferred that learning how to use memory becomes more straightforward in scenarios with larger memory.

CHAPTER 5

CONCLUSION AND FUTURE WORK

The challenge of optimal decision-making within the context of Partially Observable Markov Decision Processes (POMDPs) still remains a challenge. Addressing this, particularly for real-world scenarios with partial observability, is a complex endeavor, especially when considering problems requiring remembering long-term retention of visual observations. The relevance and potential of integrating visual external memories into reinforcement learning agents within this domain have been notably under-explored due to possible disadvantages.

However, it is hoped that this can be avoided when combined with intrinsic motivation. Moreover, the external memory can be further leveraged to interpret the memory context. As an outcome of this study, the possible advantages and disadvantages of the approaches are summarized:

LSTM Based Internal Memory:
Recurrent neural networks can be employed to establish an internal memory for RL agents. This internal memory is utilized to address POMDP tasks that demand the long-term retention of observations using LSTM Networks.

- **Advantage:** Sample efficient among other architectures.
- **Disadvantage:** Internal memory cannot be seen and analyzed.

Visual Self Memory Management (VSMM):
Agents are equipped with external memory so they can benefit themselves by interacting to learn to use a memory.

- **Advantage:** Interpretable external memory.
- **Disadvantage:** Increased observation and action dimensions, resulting in sample inefficiency due to the curse of dimensionality.

Visual Self Memory Management With Intrinsic Motivation:
Motivating agents to efficiently learn how to use memory by utilizing intrinsic motivations over the memory observation novelty/count. Due to the uneven exploration in the environment, motivating agents to hold novel/less frequent observations can improve memory use.

- **Advantage:** Interpretable external memory
- **Improvement:** Better sample efficiency (similar to LSTM-based internal memory) and better generalization capabilities.

In this study, the presented approaches are analyzed and compared through ablation studies across different environments. To conclude, the achievements derived from these experiments include:

- The application of deep RL algorithms/agents to address partially observable visual problems using the provided open-source framework.
- A demonstration that external visual observation memory mechanisms, especially the VSMM method, can serve as alternatives to internal memory-based strategies.
- A realization that including intrinsic motivation methods can enhance sample efficiency performance, allowing it to match or even surpass internal memory-based methods.
- The acknowledgment that, when considering factors such as interpretability, safety, stability, and generalization capability, the VSMM method presents numerous advantages over its internal memory-based counterparts.

Lastly, the main distinctions of the proposed intrinsic motivation architectures are as follows: The frequency-based IM often demonstrates greater sample efficiency than the autoencoder-based approach. However, a significant drawback of this architecture is its potential failure in the face of noisy or continuous observations. This is because it cannot handle an infinite number of observations, given that the hashing algorithm inherently makes hashes discrete and finite.

Furthermore, in long training experiments, as the observation visitation frequencies might shift based on the exploration policy itself, these transient exploration policies can unpredictably modify the intrinsic reward. This unpredictability can lead to learning instabilities. During our experiments, minor instability effects were observed due to this issue, but all of the experiment runs achieved the overall convergence to the optimal policy.

These limitations can be addressed using autoencoder-based intrinsic motivation functions. By employing autoencoders, the intrinsic motivation function can be readily scaled across various environments. Moreover, this function excels in scenarios with noisy or continuous observation spaces. When provided with an appropriate learning rate and after processing infinitely many observation samples from the environment, the autoencoder network converges to an average reconstruction error value. As the intrinsic reward is based on the extent to which a given observation deviates from this average regarding reconstruction error, it's inferred that this intrinsic reward exhibits a decaying or vanishing effect over time during training. This phenomenon was observed to enhance overall learning stability in our experiments.

For future works, combining external memory-based approaches with safe and explainable reinforcement learning methods could open new avenues for research in safe and robust learning. Additionally, as AI systems increasingly integrate into real-world applications, the demand for transparent decision-making processes grows. Incorporating explainability into external memory-based RL approaches not only enhances their interpretability but also fosters trust among users. Furthermore, a harmonious blend of safety and memory mechanisms could lead to RL agents that are not only more capable in complex environments but also less prone to unexpected or harmful behaviors. This becomes especially crucial in mission-critical scenarios, such as healthcare or autonomous driving, where the stakes are high. Exploring this convergence could pave the way for more reliable and human-centric AI solutions.

REFERENCES

[1] Shvechikov Pave, “Partially observable markov decision process in reinforcement learning.” URL: `https://bayesgroup.github.io/bmml_sem/2018/Shvechikov_Partially%20Observable%20Markov%20Decision%20Process%20in%20Reinforcement%20Learning.pdf`, 4 2018.

[2] “Markov decision process - Wikipedia — en.wikipedia.org.” `https://en.wikipedia.org/wiki/Markov_decision_process`. [Accessed 09-Jul-2023].

[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[4] V. H. Phung and E. J. Rhee, “A deep learning approach for classification of cloud image patches on small datasets,” *Journal of information and communication convergence engineering*, vol. 3, Sep 2018.

[5] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, “Dive into deep learning,” *arXiv preprint arXiv:2106.11342*, 2021.

[6] U. Michelucci, “An introduction to autoencoders,” *arXiv preprint arXiv:2201.03898*, 2022.

[7] R. Bellman, “A markovian decision process,” *Journal of mathematics and mechanics*, pp. 679–684, 1957.

[8] R. A. Howard, “Dynamic programming and markov processes.,” 1960.

[9] S. P. Singh and R. S. Sutton, “Reinforcement learning with replacing eligibility traces,” *Machine learning*, vol. 22, pp. 123–158, 1996.

[10] C. J. C. H. Watkins, “Learning from delayed rewards,” 1989.

[11] M. T. Spaan, “Partially observable markov decision processes,” in *Reinforcement learning: State-of-the-art*, pp. 387–414, Springer, 2012.

[12] S. D. Whitehead and D. H. Ballard, “Active perception and reinforcement learning,” in *Machine Learning Proceedings 1990*, pp. 179–188, Elsevier, 1990.

[13] K. J. Astrom *et al.*, “Optimal control of markov processes with incomplete state information,” *Journal of mathematical analysis and applications*, vol. 10, no. 1, pp. 174–205, 1965.

[14] A. K. McCallum, *Reinforcement learning with selective perception and hidden state*. University of Rochester, 1996.

[15] W. S. Lovejoy, “A survey of algorithmic methods for partially observed markov decision processes,” *Annals of Operations Research*, vol. 28, no. 1, pp. 47–65, 1991.

[16] L. Peshkin, N. Meuleau, and L. Kaelbling, “Learning policies with external memory,” *arXiv preprint cs/0103003*, 2001.

[17] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, “Planning and acting in partially observable stochastic domains,” *Artificial Intelligence*, vol. 101, no. 1, pp. 99–134, 1998.

[18] M. L. Littman, “Markov games as a framework for multi-agent reinforcement learning,” in *Machine learning proceedings 1994*, pp. 157–163, Elsevier, 1994.

[19] R. T. Icarte, R. Valenzano, T. Q. Klassen, P. Christoffersen, A. massoud Farahmand, and S. A. McIlraith, “The act of remembering: A study in partially observable reinforcement learning,” 2021.

[20] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, “Dive into deep learning,” 2023.

[21] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C. Chen, and S. S. Iyengar, “A survey on deep learning: Algorithms, techniques, and applications,” *ACM Comput. Surv.*, vol. 51, sep 2018.

[22] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, Dec. 1943.

[23] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[24] “Convolution - Wikipedia — en.wikipedia.org.” https://en.wikipedia.org/wiki/Convolution. [Accessed 29-Jul-2023].

[25] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, pp. 1735–1780, Nov 1997.

[26] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” 2014.

[27] C. Olah, “Understanding lstm networks,” 2015.

[28] D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group, *Parallel distributed processing: Explorations in the microstructure of cognition, Vol. 1: Foundations*. MIT press, 1986.

[29] D. Bank, N. Koenigstein, and R. Giryes, “Autoencoders,” *Machine Learning for Data Science Handbook: Data Mining and Knowledge Discovery Handbook*, pp. 353–374, 2023.

[30] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.

[31] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” *arXiv preprint arXiv:1511.05952*, 2015.

[32] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust region policy optimization,” in *International conference on machine learning*, pp. 1889–1897, PMLR, 2015.

[33] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, “Dueling network architectures for deep reinforcement learning,” in *International conference on machine learning*, pp. 1995–2003, PMLR, 2016.

[34] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.

[35] R. M. Ryan and E. L. Deci, "Intrinsic and extrinsic motivations: Classic definitions and new directions," *Contemporary Educational Psychology*, vol. 25, pp. 54–67, 2000.

[36] A. Aubret, L. Matignon, and S. Hassas, "An information-theoretic perspective on intrinsic motivation in reinforcement learning: A survey," *Entropy*, vol. 25, no. 2, p. 327, 2023.

[37] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[38] A. K. McCallum *et al.*, "Learning to use selective attention and short-term memory in sequential tasks," in *From animals to animats 4: proceedings of the fourth international conference on simulation of adaptive behavior*, vol. 4, p. 315, MIT Press Cambridge, 1996.

[39] L.-J. Lin and T. M. Mitchell, "Reinforcement learning with hidden states," *From animals to animats*, vol. 2, pp. 271–280, 1993.

[40] B. Bakker, "Reinforcement learning with long short-term memory," *Advances in neural information processing systems*, vol. 14, 2001.

[41] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," in *2015 aaai fall symposium series*, 2015.

[42] J. Oh, V. Chockalingam, H. Lee, *et al.*, "Control of memory, active perception, and action in minecraft," in *International conference on machine learning*, pp. 2790–2799, PMLR, 2016.

[43] M. Zhang, Z. McCarthy, C. Finn, S. Levine, and P. Abbeel, "Learning deep neural network policies with continuous memory states," in *2016 IEEE international conference on robotics and automation (ICRA)*, pp. 520–527, IEEE, 2016.

[44] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *arXiv preprint arXiv:1410.5401*, 2014.

[45] W. Zaremba and I. Sutskever, "Reinforcement learning neural turing machines-revised," *arXiv preprint arXiv:1505.00521*, 2015.

[46] E. Parisotto and R. Salakhutdinov, "Neural map: Structured memory for deep reinforcement learning," *arXiv preprint arXiv:1702.08360*, 2017.

[47] A. Demir, "Learning what to memorize: Using intrinsic motivation to form useful memory in partially observable reinforcement learning," *Applied Intelligence*, pp. 1–19, 2023.

[48] A. Khan, C. Zhang, N. Atanasov, K. Karydis, V. Kumar, and D. D. Lee, "Memory augmented control networks," *arXiv preprint arXiv:1709.05706*, 2017.

[49] C.-C. Hung, T. Lillicrap, J. Abramson, Y. Wu, M. Mirza, F. Carnevale, A. Ahuja, and G. Wayne, "Optimizing agent behavior over long time scales by transporting value," *Nature communications*, vol. 10, no. 1, p. 5223, 2019.

[50] D. Jo, S. Kim, D. Nam, T. Kwon, S. Rho, J. Kim, and D. Lee, "Leco: Learnable episodic count for task-specific intrinsic reward," *Advances in Neural Information Processing Systems*, vol. 35, pp. 30432–30445, 2022.

[51] M. C. Machado, M. G. Bellemare, and M. Bowling, "Count-based exploration with the successor representation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 5125–5133, 2020.

[52] J. Martin, S. N. Sasikumar, T. Everitt, and M. Hutter, "Count-based exploration in feature space for reinforcement learning," *arXiv preprint arXiv:1706.08090*, 2017.

[53] G. Ostrovski, M. G. Bellemare, A. Oord, and R. Munos, "Count-based exploration with neural density models," in *International conference on machine learning*, pp. 2721–2730, PMLR, 2017.

[54] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," *Advances in neural information processing systems*, vol. 29, 2016.

[55] H. Tang, R. Houthooft, D. Foote, A. Stooke, O. Xi Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel, "# exploration: A study of count-based exploration

for deep reinforcement learning," *Advances in neural information processing systems*, vol. 30, 2017.

[56] T. Zhang, H. Xu, X. Wang, Y. Wu, K. Keutzer, J. E. Gonzalez, and Y. Tian, "Bebold: Exploration beyond the boundary of explored regions," *arXiv preprint arXiv:2012.08621*, 2020.

[57] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," *arXiv preprint arXiv:1810.12894*, 2018.

[58] M. Kubovčík, I. Dirgová Luptáková, and J. Pospíchal, "Signal novelty detection as an intrinsic reward for robotics," *Sensors*, vol. 23, no. 8, p. 3985, 2023.

[59] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *International conference on machine learning*, pp. 2778–2787, PMLR, 2017.

[60] M. Seurin, F. Strub, P. Preux, and O. Pietquin, "Don't do what doesn't matter: Intrinsic motivation with action usefulness," *arXiv preprint arXiv:2105.09992*, 2021.

[61] B. Bakker, "Reinforcement learning with long short-term memory," in *Advances in Neural Information Processing Systems* (T. Dietterich, S. Becker, and Z. Ghahramani, eds.), vol. 14, MIT Press, 2001.

[62] Wikipedia contributors, "Stigmergy — Wikipedia, the free encyclopedia," 2023. [Online; accessed 31-July-2023].

[63] C. Zauner, "Implementation and benchmarking of perceptual image hash functions," 2010.

[64] D. Marr and E. Hildreth, "Theory of edge detection," *Proceedings of the Royal Society of London. Series B. Biological Sciences*, vol. 207, no. 1167, pp. 187–217, 1980.

[65] R. W. Hamming, "Error detecting and error correcting codes," *The Bell system technical journal*, vol. 29, no. 2, pp. 147–160, 1950.

[66] N. Bougie and R. Ichise, “Fast and slow curiosity for high-level exploration in reinforcement learning,” *Applied Intelligence*, vol. 51, no. 2, pp. 1086–1107, 2021.

[67] M. Chevalier-Boisvert, B. Dai, M. Towers, R. de Lazcano, L. Willems, S. Lahlou, S. Pal, P. S. Castro, and J. Terry, “Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks,” *CoRR*, vol. abs/2306.13831, 2023.

[68] M. Pleines, M. Pallasch, F. Zimmer, and M. Preuss, “Memory gym: Partially observable challenges to memory-based agents,” in *International Conference on Learning Representations*, 2023.

[69] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual.* Scotts Valley, CA: CreateSpace, 2009.

[70] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan, and I. Stoica, “RLlib: Abstractions for distributed reinforcement learning,” in *International Conference on Machine Learning (ICML)*, 2018.

[71] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

APPENDIX A

EXTENDED EXPERIMENT RESULTS

This appendix chapter includes extended experimental results for additional analysis and performance results in Figures A.1 - A.17.

Memory Environment Results:

First, the results are presented in terms of both average and individual performances. To highlight the effects of learning stability, individual outcomes for each run are also provided. Subsequently, the figures from the generalization scenario experiments are showcased. Finally, figures detailing changes in entropy, learning throughput, and loss value comparisons are shared. In Tables A.1, A.2, and A.3, methods for evaluating and comparing metrics among the algorithms are presented.

Figure and Metric Name	How to evaluate?
Figure A.1 - LSTM Algorithm: Average total cumulative rewards per episode.	Higher value and lower variance is better.
Figure A.2- SMM Algorithm: Average total cumulative rewards per episode.	Higher value and lower variance is better.
Figure A.3 - SMM Algorithm With Frequency Based IM: Average total cumulative rewards per episode.	Higher value and lower variance is better.
Figure A.4 - SMM Algorithm With Autoencoder Based IM: Average total cumulative rewards per episode.	Higher value and lower variance is better.
Figure A.5 - SMM Algorithm With Longer Memory: Average total cumulative rewards per episode.	Higher value and lower variance is better.

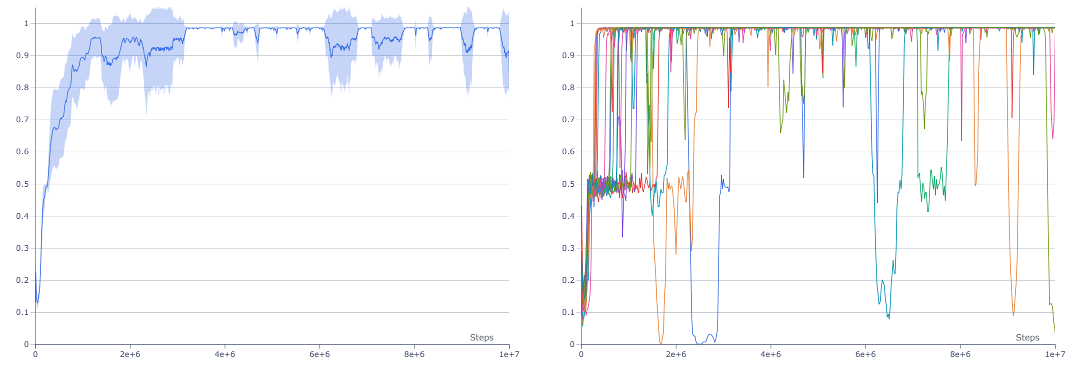
Table A.1: Extended summary of Memory Environment ablation studies.

Figure and Metric Name	How to evaluate?
Figure A.6 - Generalization Scenario: Comparison of average episode lengths per episode.	Lower value and lower variance is better.
Figure A.7 - Generalization Scenario: Comparison of average length of unchanged memory sequences per episode.	Higher value and lower variance is better.
Figure A.8 - Generalization Scenario: Comparison of average number of memory updates per episode.	Lower value and lower variance is better.

Table A.2: Extended summary of generalization scenario experiments.

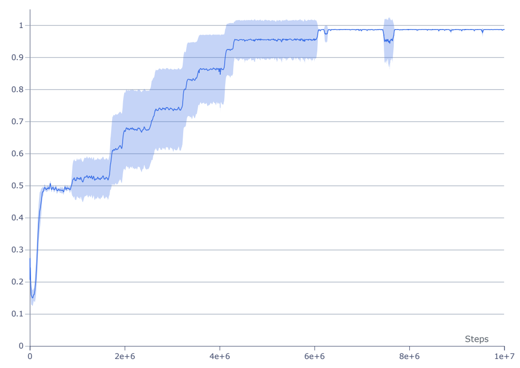
Figure and Metric Name	**How to evaluate?**
Figure A.9 - Average Policy Entropy Values Comparision.	Lower entropy means less exploration and convergence of the algorithm exploration, only visualized for informative purposes..
Figure A.10 - Average Learning Throughput Values Comparision.	Higher value and lower variance is better.
Figure A.11 - Average Total Loss Values Comparision.	Lower value and lower variance is better.

Table A.3: Extended summary of other experiments and utility metrics.

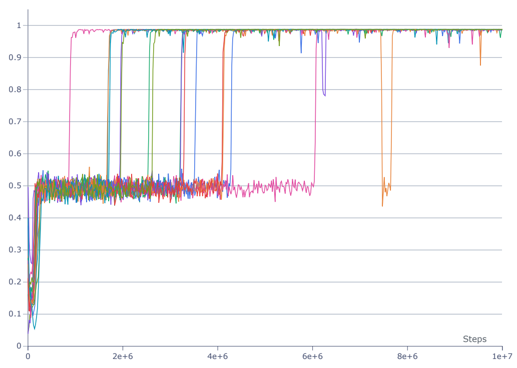


(a) Average of 16 runs with confidence intervals. (b) Individual performance of 16 parallel runs.

Figure A.1: Memory Environment - LSTM Algorithm: Average total cumulative rewards per episode.

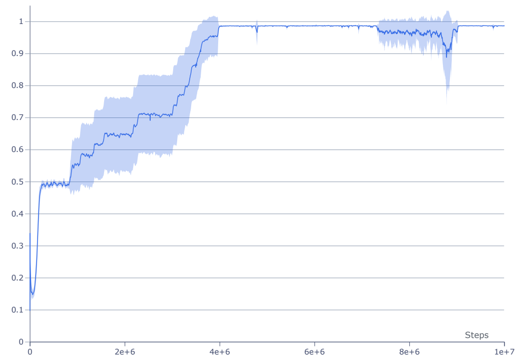


(a) Average of 16 runs with confidence intervals.

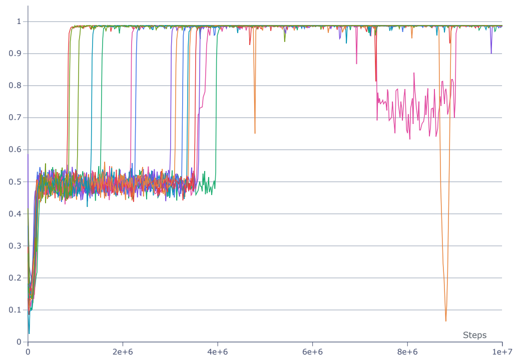


(b) Individual performance of 16 parallel runs.

Figure A.2: Memory Environment - SMM Algorithm: Average total cumulative rewards per episode.

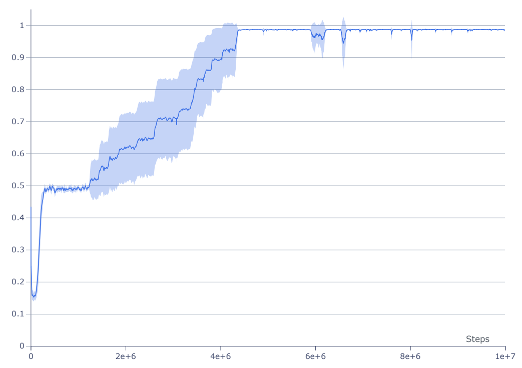


(a) Average of 16 runs with confidence intervals.



(b) Individual performance of 16 parallel runs.

Figure A.3: Memory Environment - SMM Algorithm With Frequency Based IM: Average total cumulative rewards per episode.


(a) Average of 16 runs with confidence intervals.

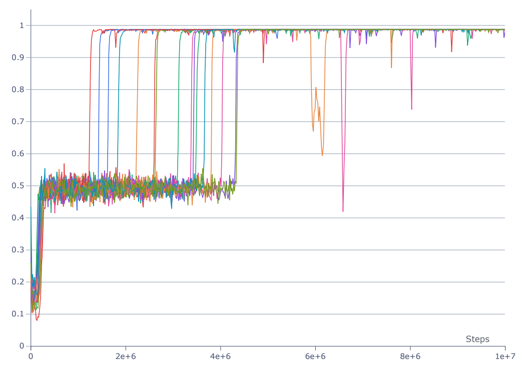
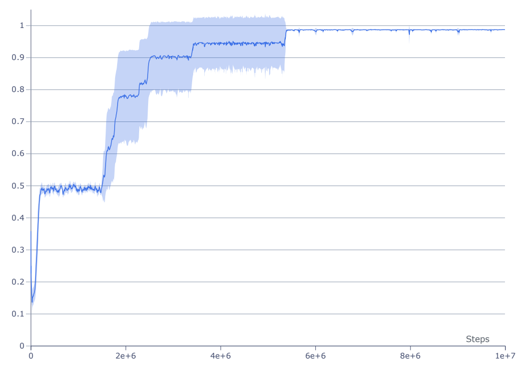

(b) Individual performance of 16 parallel runs.

Figure A.4: Memory Environment - SMM Algorithm With Autoencoder Based IM: Average total cumulative rewards per episode.


(a) Average of 12 runs with confidence intervals.

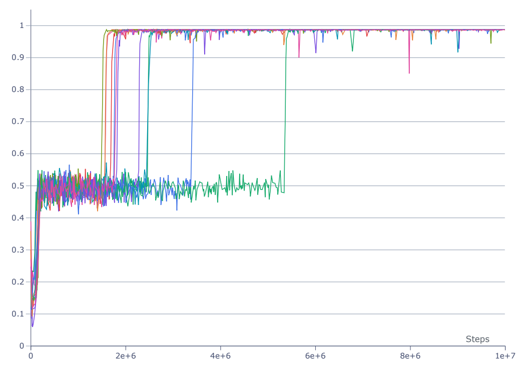
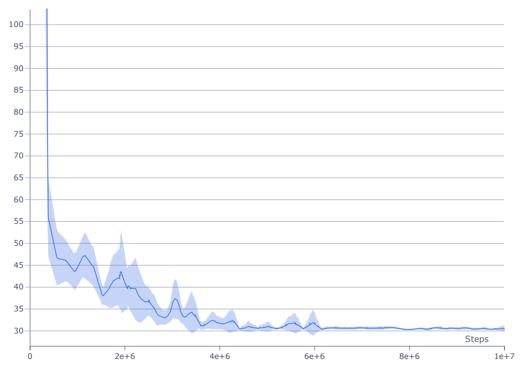
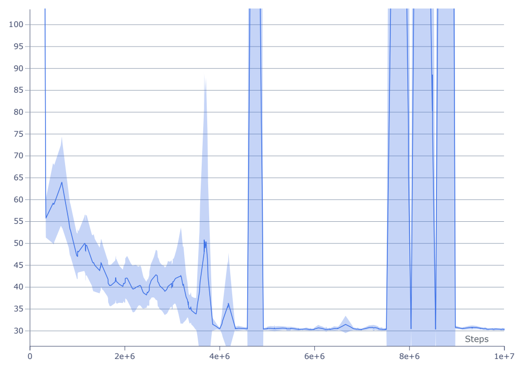

(b) Individual performance of 12 parallel runs.

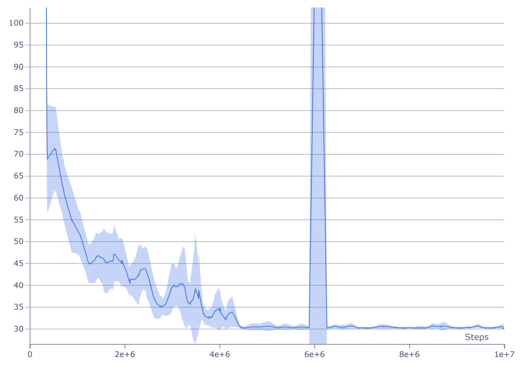
Figure A.5: Memory Environment - SMM Algorithm With Longer Memory: Average total cumulative rewards per episode.



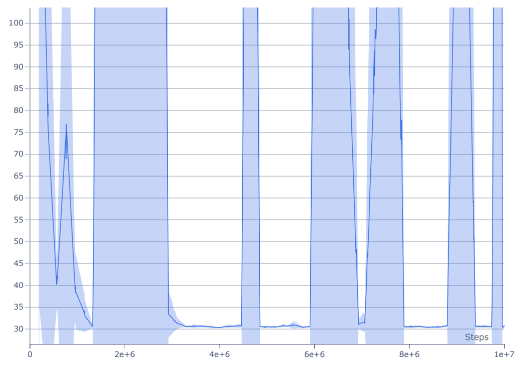
(a) VSMM



(b) VSMM With Frequency Based IM

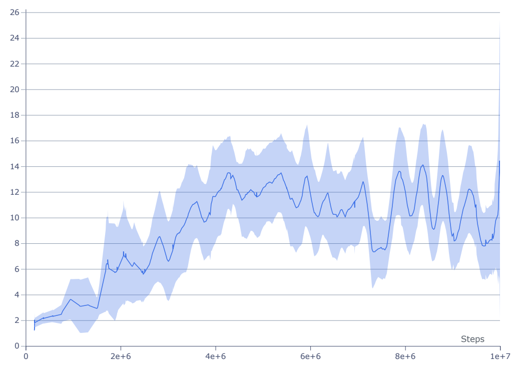


(c) VSMM With Autoencoder Based IM

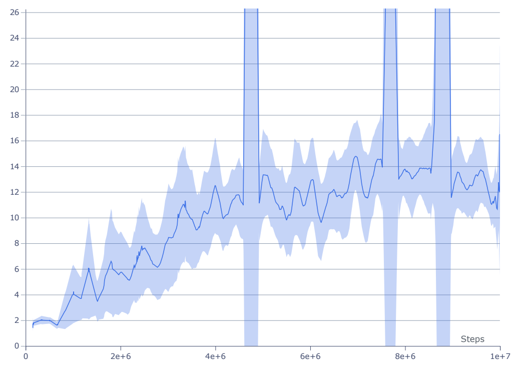


(d) LSTM

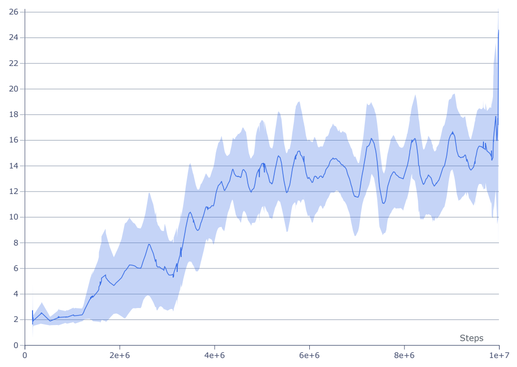
Figure A.6: Memory Environment - Generalization Scenario: Comparison of average episode lengths per episode. Each line represents the average of 16 parallel runs with confidence intervals.



(a) VSMM

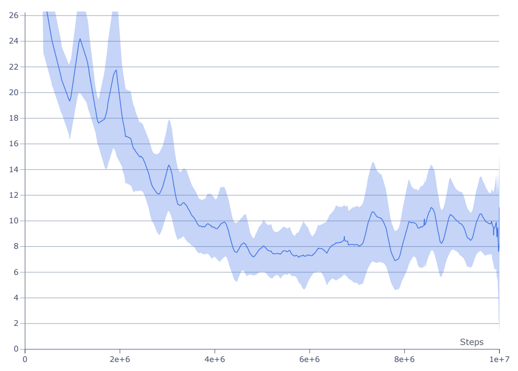


(b) VSMM With Frequency Based IM

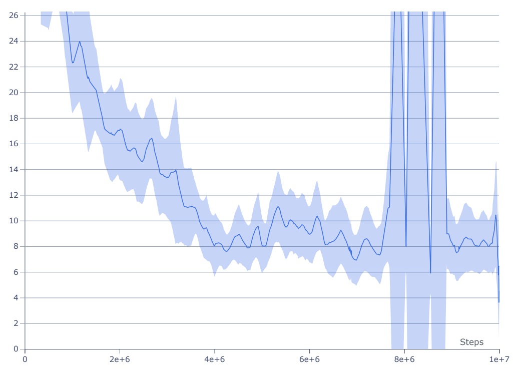


(c) VSMM With Autoencoder Based IM

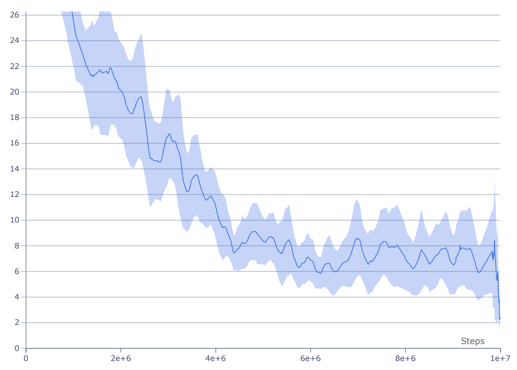
Figure A.7: Memory Environment - Generalization Scenario: Comparison of average length of unchanged memory sequences per episode. Each line represents the average of 16 parallel runs with confidence intervals.



(a) VSMM

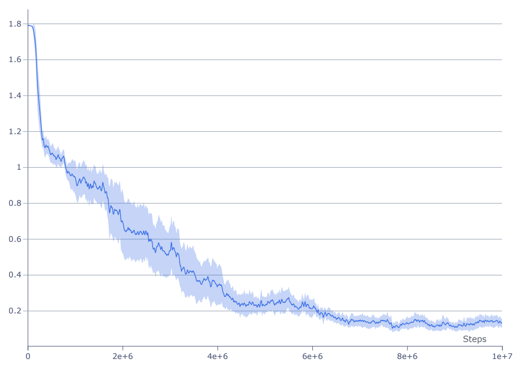


(b) VSMM With Frequency Based IM

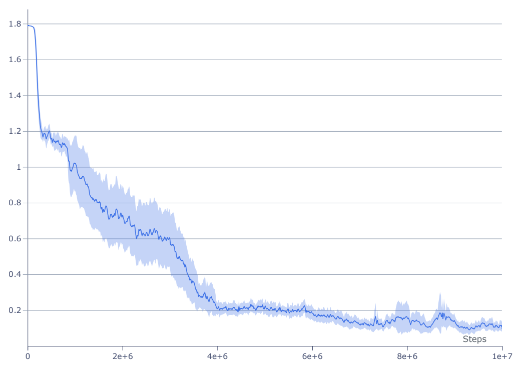


(c) VSMM With Autoencoder Based IM

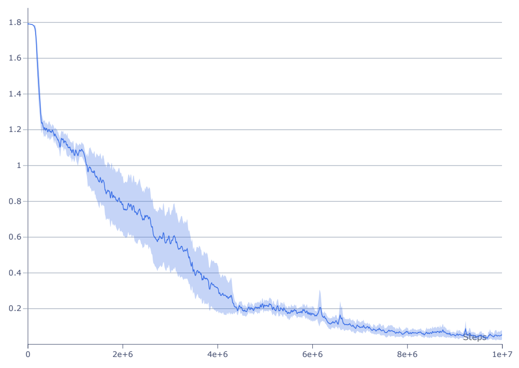
Figure A.8: Memory Environment - Generalization Scenario: Comparison of average number of memory updates per episode. Each line represents the average of 16 parallel runs with confidence intervals.



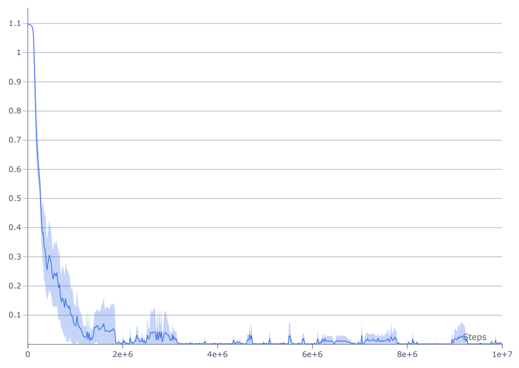
(a) VSMM



(b) VSMM With Frequency Based IM

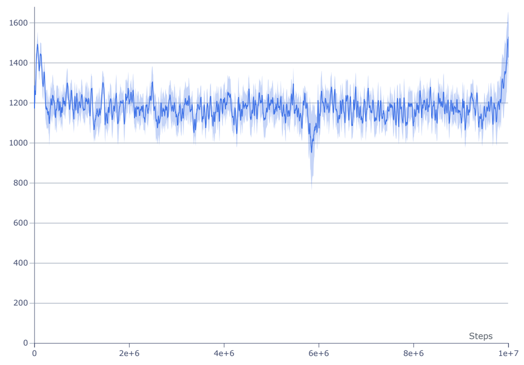


(c) VSMM With Autoencoder Based IM

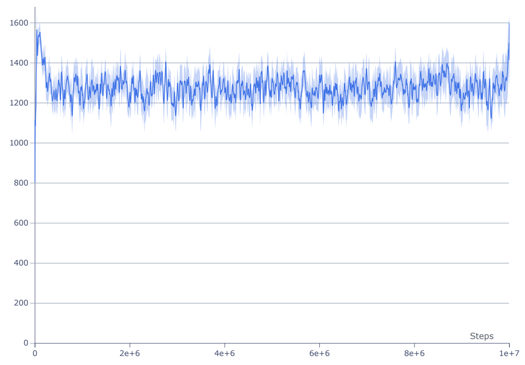


(d) LSTM

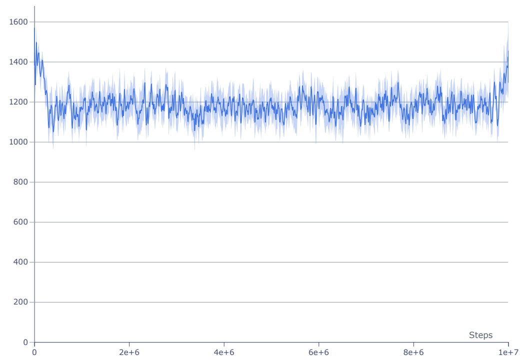
Figure A.9: Memory Environment - Average Policy Entropy Values Comparision. Each line represents the average of 16 parallel runs with confidence intervals.



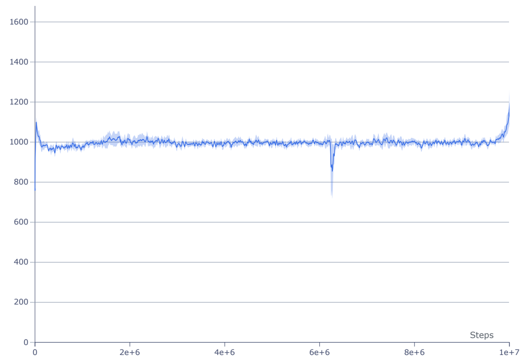
(a) VSMM



(b) VSMM With Frequency Based IM

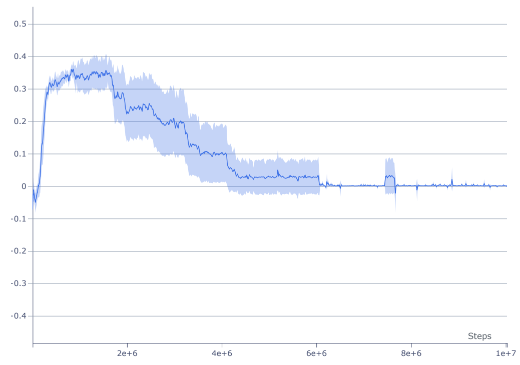


(c) VSMM With Autoencoder Based IM

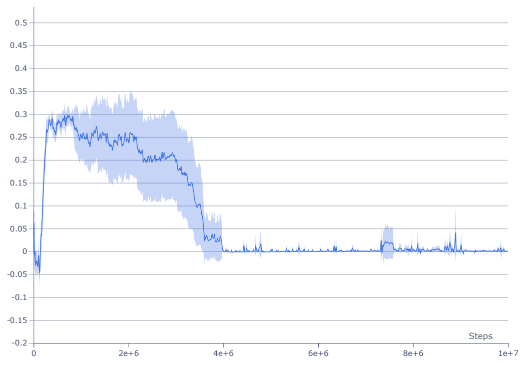


(d) LSTM

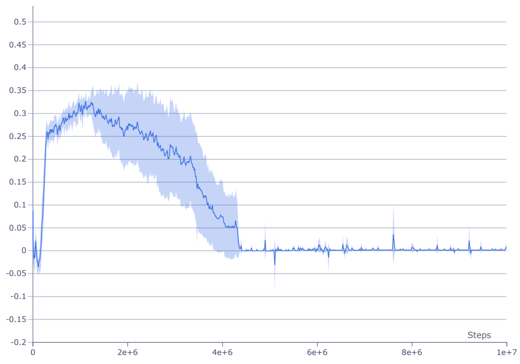
Figure A.10: Memory Environment - Average Learning Throughput Values Comparision. Each line represents the average of 16 parallel runs with confidence intervals.



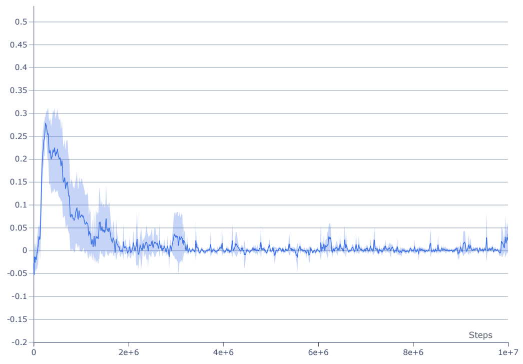
(a) VSMM



(b) VSMM With Frequency Based IM



(c) VSMM With Autoencoder Based IM



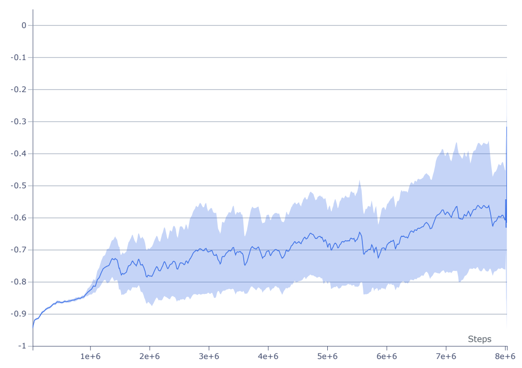
(d) LSTM

Figure A.11: Memory Environment - Average Total Loss Values Comparision. Each line represents the average of 16 parallel runs with confidence intervals.

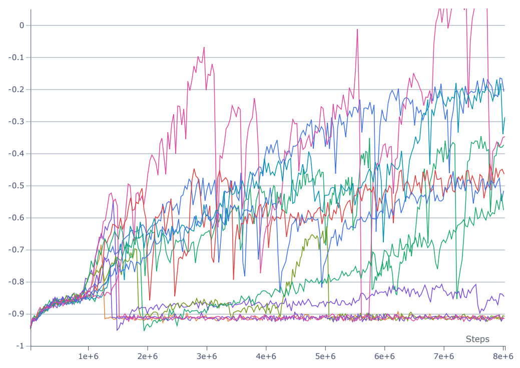
MMG Environment Results:

Figure and Metric Name	**How to evaluate?**
Figure A.12 - LSTM Algorithm: Average total cumulative rewards per episode.	Higher value and lower variance is better.
Figure A.13 - SMM Algorithm: Average total cumulative rewards per episode.	Higher value and lower variance is better.
Figure A.14 - SMM Algorithm With Frequency Based IM: Average total cumulative rewards per episode.	Higher value and lower variance is better.
Figure A.15 - SMM Algorithm With Autoencoder Based IM: Average total cumulative rewards per episode.	Higher value and lower variance is better.
Figure A.16 - Average Policy Entropy Values Comparision.	Lower entropy means less exploration and convergence of the algorithm exploration, only visualized for informative purposes.
Figure A.17 - Average Learning Throughput Values Comparision.	Higher value and lower variance is better.

Table A.4: Extended summary of MMG Environment ablation studies and other utility metrics.

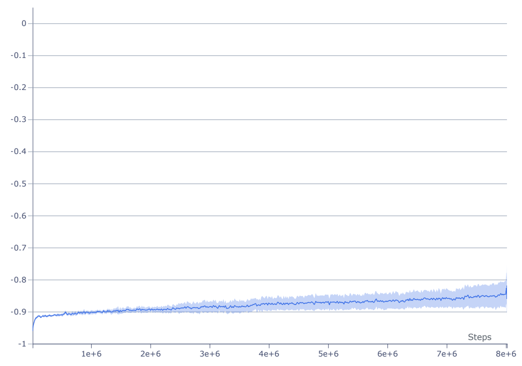


(a) Average of 12 runs with confidence intervals.

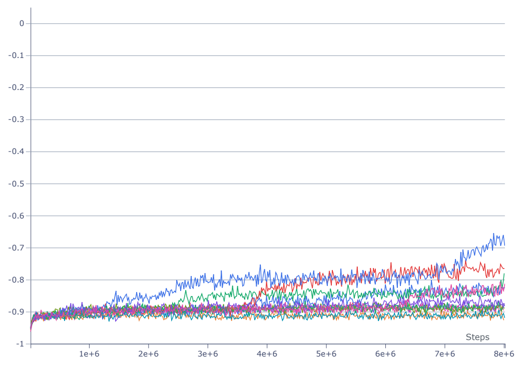


(b) Individual performance of 12 parallel runs.

Figure A.12: MMG Environment - LSTM Algorithm: Average total cumulative rewards per episode.

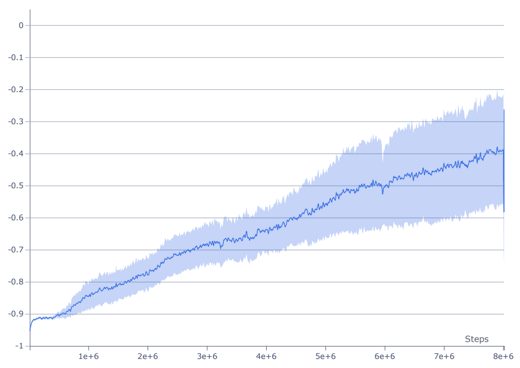


(a) Average of 12 runs with confidence intervals.

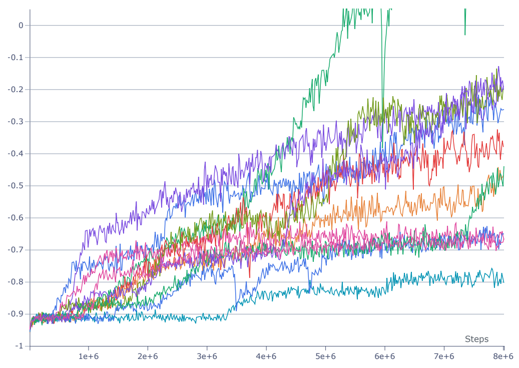


(b) Individual performance of 12 parallel runs.

Figure A.13: MMG Environment - SMM Algorithm: Average total cumulative rewards per episode.

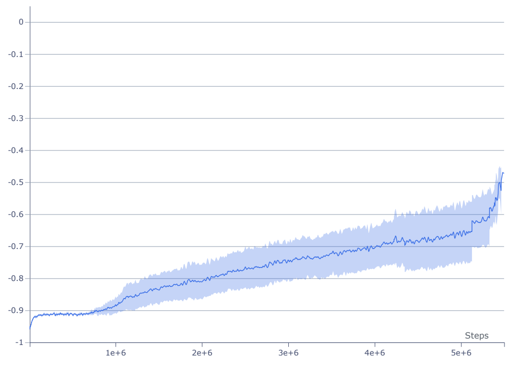


(a) Average of 12 runs with confidence intervals.

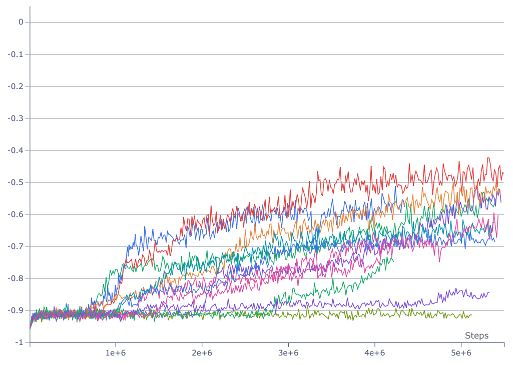


(b) Individual performance of 12 parallel runs.

Figure A.14: MMG Environment - SMM Algorithm With Frequency Based IM: Average total cumulative rewards per episode.

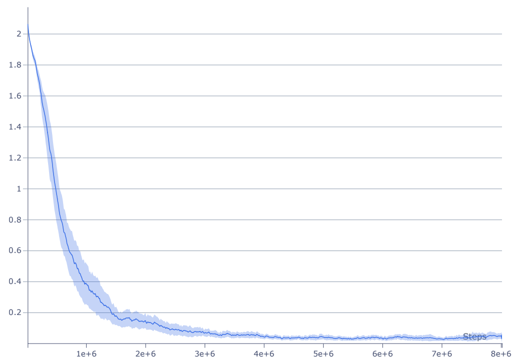


(a) Average of 12 runs with confidence intervals.

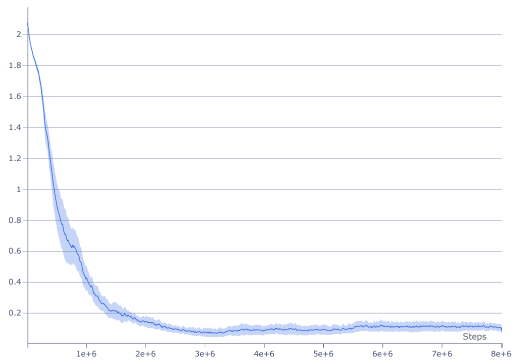


(b) Individual performance of 12 parallel runs.

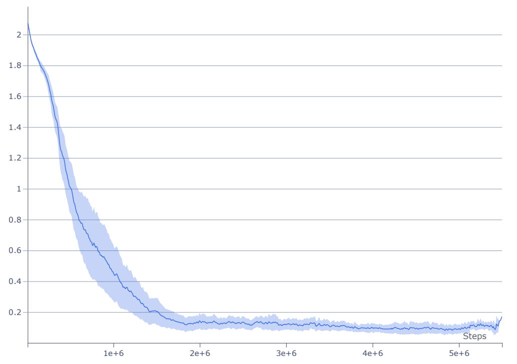
Figure A.15: MMG Environment - SMM Algorithm With Autoencoder Based IM: Average total cumulative rewards per episode.



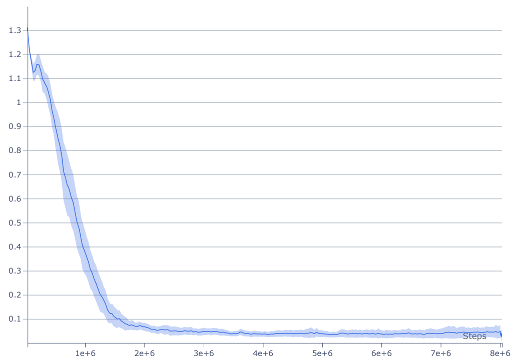
(a) VSMM



(b) VSMM With Frequency Based IM

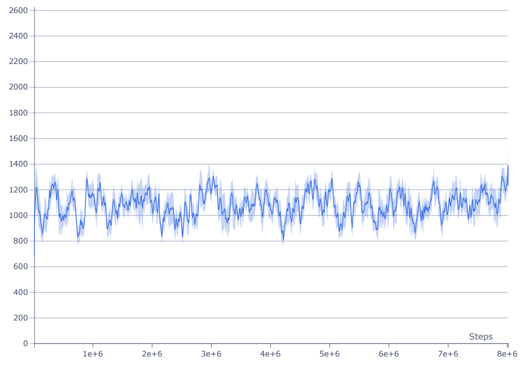


(c) VSMM With Autoencoder Based IM

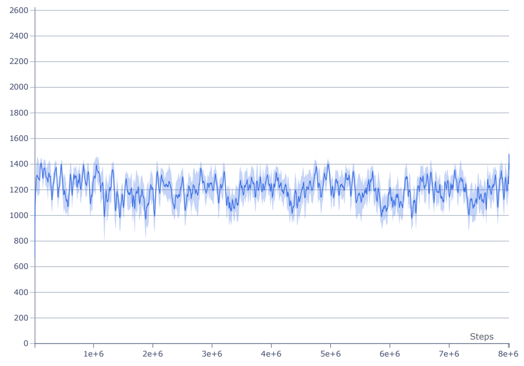


(d) LSTM

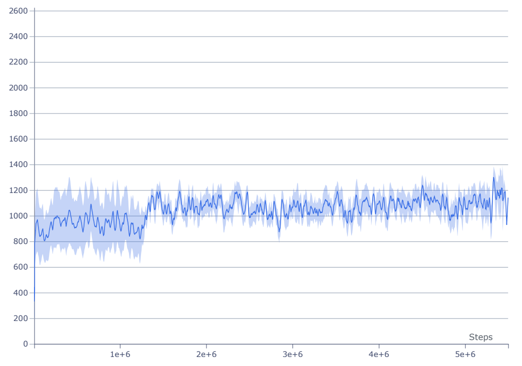
Figure A.16: MMG Environment - Average Policy Entropy Values Comparision. Each line represents the average of 12 parallel runs with confidence intervals.



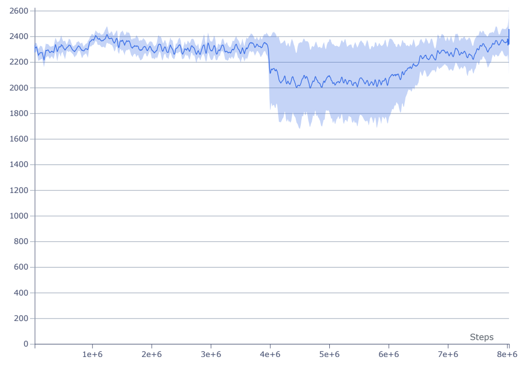
(a) VSMM



(b) VSMM With Frequency Based IM



(c) VSMM With Autoencoder Based IM



(d) LSTM

Figure A.17: MMG Environment - Average Learning Throughput Values Comparision. Each line represents the average of 12 parallel runs with confidence intervals.