Memory-based Deep Reinforcement Learning in Endless Imperfect Information Games

Dissertation

zur Erlangung des Grades eines

Doktors der Ingenieurwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik
von

Marco Pleines

Dortmund

2023

Tag der mündlichen Prüfung:	**11.12.2023**
Dekan	**Prof. Dr.-Ing. Gernot A. Fink**
Gutachter	**Prof. Dr. Günter Rudolph** (TU Dortmund, Fakultät für Informatik)
	Prof. Dr. Mike Preuss (Universiteit Leiden, LIACS)

Abstract

Memory capabilities in Deep Reinforcement Learning (DRL) agents have become increasingly crucial, especially in tasks characterized by partial observability or imperfect information. However, the field faces two significant challenges: the absence of a universally accepted benchmark and limited access to open-source baseline implementations.

We present "Memory Gym", a novel benchmark suite encompassing both finite and endless versions of the Mortar Mayhem, Mystery Path, and Searing Spotlights environments. The finite tasks emphasize strong dependencies on memory and memory interactions, while the remarkable endless tasks, inspired by the game "I packed my bag", act as an automatic curriculum, progressively challenging an agent's retention and recall capabilities.

To complement this benchmark, we provide two comprehensible and open-source baselines anchored on the widely-adopted Proximal Policy Optimization algorithm. The first employs a recurrent mechanism through a Gated Recurrent Unit (GRU) cell, while the second adopts an attention-based approach using Transformer-XL (TrXL) for episodic memory with a sliding window. Given the dearth of readily available transformer-based DRL implementations, our TrXL baseline offers significant value.

Our results reveal an intriguing performance dynamic: TrXL is often superior in finite tasks, but in the endless environments, GRU unexpectedly marks a comeback. This discrepancy prompts further investigation into TrXL's potential limitations, including whether its initial query misses temporal cues, the impact of stale hidden states, and the intricacies of positional encoding.

Keywords: Memory-based Agents, Deep Reinforcement Learning, Benchmarking, Transformer-XL, Gated Recurrent Unit.

List of Underlying Publications

The following list enumerates the underlying publications of this thesis. The publications are sorted by their year of publication.

2022
Marco Pleines, Matthias Pallasch, Frank Zimmer, and Mike Preuss.
Generalization, Mayhems and Limits in Recurrent Proximal Policy Optimization.
ArXiv preprint. CoRR, abs/2205.11104.
`https://doi.org/10.48550/arXiv.2205.11104`.

2023
Marco Pleines, Matthias Pallasch, Frank Zimmer, and Mike Preuss.
Memory Gym: Partially Observable Challenges to Memory-Based Agents.
In *The Eleventh International Conference on Learning Representations*.
`https://openreview.net/forum?id=jHc8dCx6DDr`.

2023
Marco Pleines, Matthias Pallasch, Frank Zimmer, and Mike Preuss.
Memory Gym: Partially Observable Challenges to Memory-Based Agents in Endless Episodes.
ArXiv preprint. Under Review at JMLR. CoRR, abs/2309.17207.
`https://doi.org/10.48550/arXiv.2309.17207`.

The subsequent list enumerates publications that are not considered in the underlying thesis, but emerged during the time of writing.

2020
Marco Pleines, Jenia Jitsev, Mike Preuss, and Frank Zimmer.
Obstacle Tower Without Human Demonstrations: How Far a Deep Feed-Forward Network Goes with Reinforcement Learning.
In *IEEE Conference on Games (CoG)*, pages 447-454.
`https://doi.org/10.1109/CoG47356.2020.9231802`.
Best Paper Candidate.

2022
Marco Pleines, Konstantin Ramthun, Yannik Wegener, Hendrik Meyer, Matthias Pallasch, Sebastian Prior, Jannik Drögemüller, Leon Büttinghaus, Thilo Röthemeyer, Alexander Kaschwig, Oliver Chmurzynski, Frederik Rohkrähmer, Roman Kalkreuth, Frank Zimmer, and Mike Preuss.
On the Verge of Solving Rocket League using Deep Reinforcement Learning and Sim-to-sim Transfer.
In *IEEE Conference on Games (CoG)*, pages 253-260.
`https://doi.org/10.1109/CoG51982.2022.9893628`.

2022
Jonas Schumacher and Marco Pleines.
Improving Bidding and Playing Strategies in the Trick-Taking game Wizard using Deep Q-Networks.
In *IEEE Conference on Games (CoG)*, pages 307-314.
`https://doi.org/10.1109/CoG51982.2022.9893614`.

Table of Contents

Chapter 1

Introduction

In the rapidly evolving landscape of Deep Reinforcement Learning (DRL), the role of memory in decision-making agents is becoming paramount. Real-world tasks often present agents with imperfect or partial information, where the ability to recall past observations can be the difference between success and failure. For instance, in financial trading, an agent might benefit from remembering price trends over time to make informed decisions. Similarly, in autonomous driving, a car might need to remember the recent actions of nearby vehicles to predict their future movements and navigate safely.

Recognizing the importance of memory in such scenarios, our research introduces Memory Gym, a benchmark specifically designed to assess the memory capabilities of agents in endless tasks. Drawing inspiration from the classic car game "I packed my bag", this benchmark challenges agents to remember and act upon an ever-growing list of items. At the heart of our contribution are two memory-driven baseline implementations for Proximal Policy Optimization (PPO) (Schulman et al., 2017): one powered by Transformer-XL (Vaswani et al., 2017) and the other by Gated Recurrent Unit (GRU) (Cho et al., 2014). These baselines make memory-dependent agents more accessible to the research community, paving the way for further exploration in this crucial area.

This introduction sets the stage for our key contributions, emphasizing the critical role of memory in decision-making agents, especially when faced with imperfect information. We conclude this section by providing an overview of the structure and flow of this thesis.

1.1 Memory in the Context of Decision-Making Agents

"I packed my bag" is a classic memory game often played during car journeys. Participants take turns adding items to an imaginary bag and must remember and recite the entire list in the correct order. As the list grows, the challenge intensifies, testing the limits of each player's memory. The game's endless nature highlights the challenges of retention and recall, showcasing the varying memory capacities of individuals.

Memory is not just a game; it is a critical tool for intelligent decision-making under imperfect information and uncertainty. Without the ability to recall past experiences, reasoning, creativity, planning, and learning may become elusive.

In the realm of autonomously learning decision-making agents, the agent's memory involves maintaining a representation of previous observations, a knowledge bank that grounds its next decision. Memory mechanisms, be it through recurrent neural networks (Rumelhart et al., 1986) or transformers (Vaswani et al., 2017), have enabled these agents to master tasks both virtual and real. For instance, DRL methods have conquered complex video games as Capture the flag (Jaderberg et al., 2019), StarCraft II (Vinyals et al., 2019), and DotA 2 (Berner et al., 2019). Their success extends beyond virtual environments to real-world challenges as dexterous in-hand manipulation (Andrychowicz et al., 2020) and controlling tokamak plasmas (Degrave et al., 2022). The remarkable achievements of DRL agents empowered by memory often come coupled with substantial computational demands and additional techniques that extend beyond the scope of benchmarking an agent's memory interaction abilities.

1.2 Contribution: Endless Task Memory Benchmark

We propose Memory Gym (Pleines et al., 2023), an open-source benchmark, designed to challenge memory-based DRL agents to memorize events across long sequences, generalize, and be sample efficient. Memory Gym encompasses three unique environment families: Mortar Mayhem, Mystery Path, and Searing Spotlights. Every environment within these families is procedurally generated and offers visual observations and multi-discrete action spaces. Importantly, mastering these environments is impossible without effective memory utilization, necessitating regular memory interactions.

Memory Gym offers not only typical finite memory challenges with a single goal but remarkably extends these to endless tasks, inspired by the "I packed my bag" game. Such novel endless tasks, behaving as an automatic curriculum, are pivotal for not just evaluating efficiency but, more critically, assessing the effectiveness of memory mechanisms. As mirrored in the aforementioned game, capitulating to the ever-expanding list becomes inevitable. More effective memory mechanisms may handle lengthier lists than less effective ones.

To our current understanding, no existing Deep Learning (DL) memory benchmark possesses such endless tasks targeting sequence models. Additionally, DRL tasks exhibit dynamic behaviors due to the evolving nature of the agent's policy during training. As a result, episodes can yield vastly different outcomes. For example, episode lengths can vary significantly based on the decisions made, particularly when dealing with stochastic agents or environments. This contrasts supervised learning tasks, which typically rely on a consistent data set.

As detailed in this work, many existing memory tasks exhibit shortcomings, giving reason to the absence of a standardized memory benchmark. This underscores the importance of introducing refined tasks, as achieved by Memory Gym, which we believe to be an invaluable contribution to the field of benchmarking memory-based DRL agents.

1.3 Contribution: Memory-based Agent Baselines

A recurring theme in the realm of memory-based DRL is the lack of accessible and reproducible baselines. Many significant works, particularly those utilizing transformers (Fortunato et al., 2019; Parisotto et al., 2020; Hill et al., 2021; Lampinen et al., 2021; Parisotto and Salakhutdinov, 2021), have not made their implementations publicly available. As implementation details of DRL algorithms are vital (Engstrom et al., 2020; Andrychowicz et al., 2021; Huang et al., 2022a), this lack of transparency poses significant challenges for the research community, hindering progress and reproducibility.

In response to this challenge, we contribute two accessible and easy-to-follow baseline implementations. These are based on the widely-adopted DRL algorithm Proximal Policy Optimization (PPO) (Schulman et al., 2017). To allow for memory, one baseline is centered around a Recurrent Neural Network (RNN) leveraging a GRU cell, while the other one employs Transformer-XL (TrXL) as an episodic memory leveraging a sliding window approach.

Integrating these memory mechanisms is not as simple as plug-and-play. Their inclusion increases the implementation's complexity, necessitating the collection of more data, carefully processing of sequential data, and the expansion of agent's interfaces to its model, usually an Artificial Neural Network (ANN). Such complexities can be vulnerable to errors and inflate the hardware memory overhead. Recognizing these challenges, our baselines are designed to be intuitive and easy to adopt. They serve as a beacon for the community, especially given the dearth of readily available transformer-based DRL implementations.

In our experimental analysis, we benchmark the TrXL and GRU agents on the finite and endless tasks presented by Memory Gym. The results indicate that TrXL outperforms in terms of sample efficiency on Mystery Path and effectiveness on Mortar Mayhem. However, in the Searing Spotlights environment, GRU demonstrates refined sample efficiency. To our greatest surprise, in the endless environments, GRU consistently surpasses TrXL in effectiveness by large margins. In our later discussion, we explore five potential hypotheses to shed light on the unexpectedly low effectiveness of TrXL.

1.4 Limitations

In the pursuit of advancing the understanding of memory-based DRL, our research has made significant contributions. However, it is imperative to recognize the constraints of our study.

The underlying research was conducted within a limited computational budget, approximately 50,000 GPU hours. This constraint inevitably influenced the depth and breadth of our experiments. For example, extensive ablation studies on baselines, such as scaling the agents' models, or exploring the environments in greater depth, were not feasible within this budget. Similarly, integrating additional baselines as HELM (Paischer et al., 2022a) or state-of-the-art DRL algorithms as Soft Actor-Critic (Haarnoja et al., 2018) was beyond our reach, especially considering the associated costs of hyperparameter tuning.

While Memory Gym is a valuable benchmark, it does not claim to be a universal standard for all kinds of memory-based tasks in DRL. We focused on tasks that solely challenge memory and deliberately excluded others that comprise a broader context. For instance, while continuous actions are pivotal in many applications such as control tasks, our research did not venture into creating an environment family for them. Instead, we prioritized refining the quality of our existing tasks rather than expanding the quantity.

Our research is centered on memory within the realm of DRL. While there are benchmarks in other contexts, such as natural language processing or supervised learning (e.g. Long Range Arena (Tay et al., 2021)), our study did not encompass these areas. The vast landscape of such research would have significantly expanded the scope of our work, potentially diluting our focus on memory in DRL agents.

1.5 Overview and Structure

This thesis is divided into eight chapters to provide a comprehensive understanding of memory-based DRL.

In the subsequent chapter, we lay the foundational concepts of DRL. We trace the evolution of the core algorithm PPO, starting with its rudimentary predecessor, REINFORCE (Williams, 1992). This is further developed into Advantage Actor-Critic (A2C) and eventually refined into PPO, incorporating a clipped surrogate objective and Generalized Advantage Estimation (Schulman et al., 2016). We also delve into the intricacies of notable sequence models, including GRU, Long Short-Term Memory (LSTM), TrXL, and Gated Transformer-XL (GTrXL).

Chapter 3 reviews relevant literature, focusing on existing memory-based baselines and tasks. A significant observation is the prevalent lack of accessible and reproducible baselines. We introduce criteria essential for benchmarks, especially those targeting memory tasks. An analysis of these benchmarks reveals inconsistencies and gaps, explaining why many memory baselines do not employ a consistent set of tasks for agent assessment.

To fill in this gap, Chapter 4 unveils our memory task families: Mortar Mayhem, Mystery Path, and Searing Spotlights. We elucidate both their finite and endless versions, comparing them with existing tasks. The chapter also offers insights into scaling the challenges posed by these environments and evaluates their dependence on frequent memory interactions.

In Chapter 5, we detail our baseline contributions. We first outline the general actor-critic model architecture, which accommodates multi-discrete actions and optionally supports observation reconstruction and ground estimation targets. Afterwards, we delve into the specifics of the GRU and TrXL agents.

Chapter 6 is dedicated to experimentation, where Memory Gym's environments are pitted against the GRU and TrXL baselines. We describe the methodologies behind training and evaluation, encompassing hyperparameters, architectural nuances, and statistical tools employed. The results section reveals intriguing findings, including the unexpected superiority of GRU over TrXL in endless scenarios. We also illustrate the computational demands of our training runs and the behaviors exhibited by the resulting agents.

In Chapter 7, we discuss our most notable findings. We propose five hypotheses to potentially narrow the unexpected performance dynamics of TrXL in endless tasks. These hypotheses touch upon agent model capacity, learning signal strength, temporal information lacks in TrXL's initial query, outdated hidden states, and challenges with positional encoding. We also critically evaluate the Searing Spotlights environment and, based on our discussions, suggest potential refinements. We conclude the chapter by positioning Memory Gym in the broader context of standard benchmarks, advocating its complementary rather than exhaustive nature.

The thesis culminates with a recapitulation of our pivotal findings, followed by potential avenues for future research. We then offer a reflective critique of our work, highlighting its limitations and areas for improvement.

CHAPTER 2

Fundamentals

To follow our contributions, it is essential to understand the foundational concepts they are built upon. We begin by exploring the principles of DRL, followed by an in-depth look at the widely-used DRL algorithm PPO. Additionally, we explore recurrent and attention-based neural networks, specifically LSTM, GRU, TrXL, and GTrXL, which are renowned for tackling memory-dependent tasks.

2.1 Deep Reinforcement Learning

In the realm of machine learning, Reinforcement Learning (RL) stands alongside Supervised and Unsupervised Learning as a unique subset of data-driven methods. Unlike these methods where a pre-existing dataset is usually available, RL addresses problems of sequential decision-making, often starting without prior knowledge. Here, an entity known as the agent interacts with its surroundings to complete tasks that demand a series of decisions, or actions. The environment provides feedback in the form of rewards whenever the agent succeeds. Negative rewards can be signaled to the agent to diminish undesired behaviors and outcomes. The agent then uses this feedback to reinforce its future decision-making strategies. This strategy, which governs how the agent decides on actions based on the environment's state, is commonly referred to as a "policy". The overall goal in RL is to find an optimal policy that maximizes the rewards received by the agent.

Traditional RL algorithms usually utilize tabular data structures, where the dimensions are determined by the number of possible states in an environment and the actions an agent can execute within it. However, as environments and agents become more complex, these methods face scalability issues, stemming from the challenges of performing table look-ups or accommodating potentially boundless tables in computer memory. Merging the principles of RL with the capabilities of ANNs led to the emergence of the DRL domain. A ground breaking success in this field is the Deep Q-Network (DQN) algorithm by Mnih et al. (2015), which demonstrates human-level proficiency in playing a diverse range of Atari games.

With the context of DRL now set, the subsequent sections delve into its foundational concepts vital for introducing PPO. At first, the Markov Decision Process (MDP), the goal of the agent, the Markov property, and the Partially Observable Markov Decision

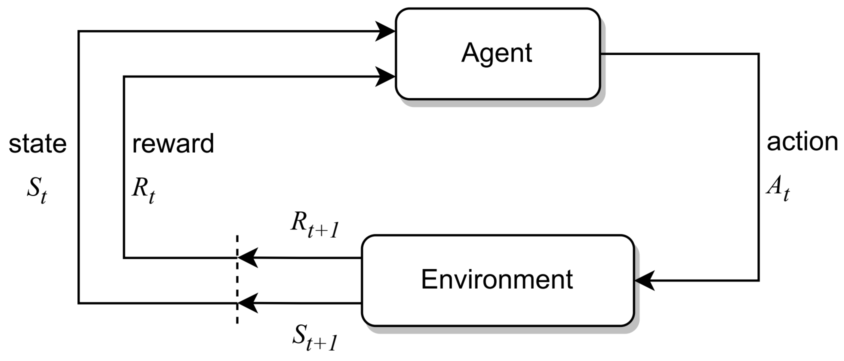


Figure 2.1: The Markov Decision Process depicts a continuous interaction loop between the agent and the environment. Once the environment's state is exposed, the agent decides on and executes an action in response. Next, the environment transitions to a new state and conveys a reward to the agent, which the agent utilizes to refine its policy. Adapted from Sutton and Barto (2018).

Process (POMDP) are detailed. This is followed by differentiating deterministic and stochastic policies. We then transition to defining both the state-value function and the action-value function, highlighting their learning processes through Monte Carlo Prediction, Temporal Difference Learning, and n-step returns. Furthermore, we distinguish DRL algorithms based on their on-policy or off-policy characteristics. Finally, we enumerate the key challenges of this domain. The exploration of theoretical foundations, such as the derivation of the policy gradient theorem and the Bellman equation, are considered out of scope. Unless otherwise specified, the notation and definitions adhere to those presented in the book "Reinforcement Learning: An Introduction" by Sutton and Barto (2018).

2.1.1 Markov Decision Process

In RL, an autonomous agent primarily learns through its interactions. This sequential decision-making is formalized by the Markov Decision Process (MDP), being the foundational framework for various algorithms, agents, and environments. As depicted in Figure 2.1, this interaction is a continuous loop.

For this thesis, the environment is considered to be stochastic. Upon initialization or "reset", the environment presents its initial state S_0 from the distribution $p(S_0)$, with $p(S_0 = s)$ being the likelihood of the state being s. With this state as reference, the agent chooses an action $A_0 \in \mathcal{A}(s)$ to execute at time step $t = 0$. The action space, $\mathcal{A}(s)$, represents possible actions when in state s.

After taking the action, the environment transitions to a new state S_{t+1} at time $t + 1$ and provides the agent with an expected reward R_{t+1}, defined by the reward function:

$$R(s, a) \doteq \mathbb{E}[R_{t+1} | S_t = s, A_t = a] \tag{2.1}$$

Running the MDP produces a trajectory of triplets that are produced during one step (i.e. one agent-environment interaction):

$$\tau = (S_0, A_0, R_1), (S_1, A_1, R_2), (S_2, A_2, R_3), \ldots \tag{2.2}$$

S, A, and R are random variables, while s, a, and r are specific instances of these variables. The environment's dynamics and response to actions is governed by the transition function. This function assigns a probability to transitioning from state s and action a to a subsequent state s' and receiving an expected reward r as:

$$p(s', r|s, a) \doteq Pr\{S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a\} \tag{2.3}$$

2.1.2 Goal: Maximizing Discounted Cumulative Rewards

Over time, the agent's goal is to strategically act in ways that maximize the received cumulative rewards. The rewards accumulated by the agent are referred to as the return G_t:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T \tag{2.4}$$

where T represents the terminal time step. An episode in the agent-environment interaction loop spans from time step 0 to T. While some tasks are episodic as previously outlined, others persist as continuing tasks without distinct terminal states. For such continuing tasks, the return G_t becomes impractical, as T may approach infinity. To address this issue, the concept of discounting is leveraged. The agent's goal is now to maximize the expected discounted return G_t:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.5}$$

Here, γ denotes the discount factor, constrained within the interval $0 \leq \gamma \leq 1$. This factor acts as a hyperparameter, chosen prior to training. A smaller discount factor makes agents prioritize immediate rewards, making them shortsighted, whereas larger values promote farsighted behaviors. For example, when set to 0, the agent's sole objective becomes maximizing R_{t+1}.

2.1.3 Markov Property

The Markov property asserts that the probability of transitioning to the next state and emitting the expected reward is independent of the prior history of states and actions (Morales, 2020):

$$p(s', r|s, a) = Pr\{S_{t+1} = s', R_{t+1} = r \mid S_t, A_t, S_{t-1}, A_{t-1}, \ldots\} \tag{2.6}$$

A state adhering to the Markov property is commonly identified as a Markov state,

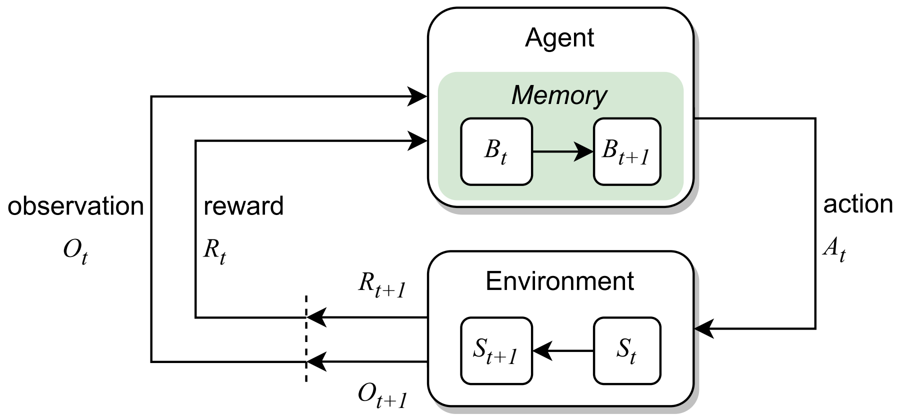


Figure 2.2: In the Partially Observable Markov Decision Process (POMDP), the agent operates in environments where the true state is hidden. The agent perceives only a partial observation, an incomplete reflection of the true state. Guided by its internal belief state and the current observation, the agent selects an action. This decision-making process updates the belief state from B_t to B_{t+1}. Subsequently, the environment transitions to a new state in response to the agent's action, providing a fresh observation and corresponding reward. We refer to the mechanism that characterizes the belief state as "memory".

characterized by its distinct representation that differentiates it from other states. Within RL, this property ensures that an agent, for optimal decision-making, does not need to remember all past states and actions. Instead, the present state, which embodies all relevant information, is sufficient. This simplifies the decision-making process, reducing the computational demands and saving computer memory. In the following section, we detail how environments with partially observable states can violate the Markov property.

2.1.4 Partially Observable Markov Decision Process

Environments can differ in the accessibility of their states. While some tasks provide complete state information, real-world problems often present scenarios where the agent can only access partial or noisy observations about the true state of the environment. This violates the Markov property as the agent needs access to the past to make meaningful decisions. The Partially Observable Markov Decision Process (POMDP) provides a formal framework to handle such scenarios.

In a POMDP, the agent does not observe or perceive the actual state S_t. Instead, it receives an observation O_t which provides imperfect information about S_t. To get an estimate on the state of the environment, the agent needs access to the history $\mathcal{H}_t$ of past observations and actions:

$$\mathcal{H}_t \doteq A_0, O_1, \ldots, A_{t-1}, O_t \tag{2.7}$$

The inherent complexity of relying on the full history $\mathcal{H}_t$ stems from its ever-growing nature as the agent interacts with the environment. Instead of using the entire history, a more tractable approach is to use the concept of a belief state.

A belief state, denoted as $b(s)$, is a probability distribution over all possible states, reflecting the agent's current belief about the environment's state given the history of observations. It provides a compact representation of the history and encapsulates all the information necessary for decision-making, thus restoring the Markov property. Formally, given a history $\mathcal{H}_t$, the belief state at time t for a state s is:

$$b_t(s) = Pr\{S_t = s|\mathcal{H}_t\} \tag{2.8}$$

The process of approximating the belief state requires what we term as "memory" within the agent. This memory distills the accumulated history into a concise representation that guides the agent's decision-making. Importantly, this representation should have the ability to retain and recall relevant information from past interactions.

Figure 2.2 illustrates the agent-environment interaction when modeled as POMDP.

2.1.5 Deterministic and Stochastic Policies

The behavior of an agent within an environment is determined by its policy, denoted as π. This policy maps states (or observations) to actions. There are primarily two types of policies: deterministic and stochastic.

A deterministic policy provides a specific action a for a state s:

$$\pi(s) = a \tag{2.9}$$

On the contrary, a stochastic policy defines a probability distribution over the possible actions for a given state s:

$$\pi(a|s) = Pr\{A_t = a \mid S_t = s\} \tag{2.10}$$

2.1.6 Value Functions

Value functions are pivotal as they estimate how good it is for an agent to be in a certain state or to perform a particular action in a state, given a policy. These estimations guide the agent's decisions in the learning process and are central to many algorithms.

The state-value function, denoted as v_π, represents the expected return from a particular state s when following the policy π:

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right] \tag{2.11}$$

The action-value function (or q-function), often termed as q_π, quantifies the expected return from taking an action a in state s and then following policy π:

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right] \tag{2.12}$$

One of the primary benefits of using the q-function is its direct application in determining actions. For a state s, the most promising action is the one that maximizes the q-value:

$$\pi(s) = \arg\max_a q_\pi(s,a) \tag{2.13}$$

This stands in contrast to the value function, where the agent may require fewer state visits but would need to anticipate future states to determine the optimal policy.

Another important value function emerges as a result of the relationship between the value function and the q-function - the so called advantage function $\hat{A}_\pi(s,a)$. The advantage function quantifies the relative benefit (or advantage) of taking a specific action a in state s. This quantity is represented by the difference between the action-value and the value function:

$$\hat{A}_\pi(s,a) = Q_\pi(s,a) - V_\pi(s) \tag{2.14}$$

Note that the capital letters $\hat{A}$, Q, and V denote array estimates instead of single scalars.

In conclusion, these value functions offer different perspectives and can be learned through experience. Common methodologies for this learning include Monte Carlo Prediction and Temporal Difference Learning, discussed in the subsequent sections. Throughout this work, we refer to the value function as the value function, while we use the term q-function for the action-value function.

2.1.7 Monte Carlo Prediction

Monte Carlo (MC) Prediction employs random sampling to estimate value functions and to offer unbiased estimations of expected returns for states or state-action pairs. The more samples provided, the closer the estimation moves to the actual value. One popular variant of this procedure is the constant-α method, which is described next.

To learn the value function V for a given policy π, the initial step is to sample a finite trajectory:

$$\tau = S_t, A_t, R_{t+1}, \ldots, R_T, S_T \sim \pi_{t:T} \tag{2.15}$$

where t typically starts at 0 and T denotes the terminal step. The subsequent step is to compute the return for each time step within the trajectory. This is accomplished by a recursive formulation of the previously defined return (Equation 2.5):

$$G_t = R_{t+1} + \gamma G_{t+1} \tag{2.16}$$

By iterating over the sampled trajectory from T backward to 0, the return for each time step is computed recursively. It is essential to note that the return for the terminal state S_T is 0 since the episode concludes and no future rewards can be acquired. With the sampled returns in hand, the value function can be incrementally updated as:

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)] \tag{2.17}$$

where $G_t - V(S_t)$ represents the MC error and α is the learning rate. For discrete state spaces, state-values are often initialized to 0 and stored in a tabular format. In more complex scenarios, universal function approximators might be used.

2.1.8 Temporal Difference Learning

Temporal Difference (TD) Learning offers a unique advantage compared to MC methods: it does not necessitate the completion of an entire episode before updating the value function. Instead, TD Learning updates the value function V after at least one step taken within the environment, a process facilitated by a technique called bootstrapping. Bootstrapping in this context refers to the practice of updating value estimates based on other estimates. For example, these estimates can be drawn from the to-be-optimized value function.

A fundamental difference between the two methods lies in their target for updates. While MC uses the sampled return G_t as its target, TD relies on an estimated target, leading to some bias but compensating with reduced variance.

The most basic form of TD learning is termed TD(0), represented by the following equation:

$$V(s_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{2.18}$$

In this equation:

- α is the learning rate,
- $R_{t+1} + \gamma V(S_{t+1})$ serves as the target, and
- the term $[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$ quantifies the TD error.

While TD(0) looks one step ahead, it can also be used to look ahead for n-steps as detailed next.

2.1.9 n-Step Returns

An n-step return serves as a bridge between MC and TD methods. While MC prediction waits until the end of the episode to compute the return, and TD(0) updates the value function based on the immediate reward and the next state's value, n-step returns provide a compromise, considering a specified number of steps ahead before bootstrapping.

To grasp this concept, first consider the two-step return. In this scenario, the agent expects two rewards before bootstrapping from the third state's value:

$$G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2}) \tag{2.19}$$

Here, $\gamma^2 V_{t+1}(S_{t+2})$ is the estimate of the terms: $\gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots + \gamma^{T-t-1} R_t$. Building on this foundation, we can generalize to the n-step return, where the subscript $t : t + n$ depicts the range from $t + 1$ to $t + n$:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}) \tag{2.20}$$

This formula captures two pivotal elements: the accrued reward over the next n steps and the bootstrapped value function from the n^{th} state. If the look-ahead exceeds the trajectory length (i.e., $t + n \geq T$), any nonexistent terms are assumed to be 0. Although n-step returns provide a trade-off, balancing bias (due to bootstrapping) and variance (from sampling), they bring about added complexity in implementation and introduce an additional hyperparameter to tune.

2.1.10 On-Policy vs. Off-Policy

RL agents can adopt one of two main strategies to learn from their environment: on-policy and off-policy. The distinction lies in the relationship between the policy used for exploration and the policy under optimization.

In on-policy learning, the agent simultaneously learns and makes decisions using its current policy. Therefore, the policy directing the agent's actions is the same policy being optimized. One implication of this approach is the inability to reuse past experiences efficiently. Since the samples derive from a previous version of the policy, they may become outdated or irrelevant for the current policy's optimization. In other words, outdated samples are considered off-policy.

Off-policy learning decouples the policy used for exploration from the policy being optimized. This means that the agent may employ different policies for exploring the environment and gathering training data. A key advantage is the capacity to learn from a wide array of experiences. This includes data gathered from past iterations of the policy, or even from an entirely different policy, such as demonstrations by a human expert.

2.1.11 Challenges in Reinforcement Learning

Developing RL agents is accompanied by a multitude of challenges that cutting edge algorithms try to overcome. Understanding and addressing these is fundamental for the advancement of RL and its application in diverse fields.

Reward Shaping: Designing an appropriate reward function can be intricate. If rewards are designed improperly, an agent might find unintended shortcuts or suboptimal behaviors. Striking a balance between guiding an agent with informative rewards and not handcrafting every detail is crucial.

Credit Assignment Problem: In environments where rewards are sparse or delayed, it becomes challenging to identify which actions were responsible for the obtained rewards. The system must trace back through its action history to assign credit (or blame) to the appropriate decisions that led to the outcome.

Exploration versus Exploitation: Agents must decide whether to explore new, potentially better strategies or exploit known strategies that work well. Striking the right balance is critical, as too much exploration can lead to inefficiency, while excessive exploitation can miss better solutions.

Generalization: An ideal RL agent should be capable of generalizing its learning to new, unseen situations. Overfitting to a specific environment or set of scenarios limits the agent's effectiveness in varied or real-world conditions.

Representation Learning: The way state and action spaces are represented can significantly impact the efficiency of learning. Developing representations that capture the essential features of the environment, while being compact and scalable, is a major challenge.

Imperfect Information: In many real-world scenarios, agents must act based on partial or noisy observations. Learning under these conditions requires the agent to develop strategies that are robust to uncertainty and possible misinformation. This intensifies the challenge of learning representations that can effectively capture, store, and retrieve past observations in a reliable manner.

Scalability: As environments become more complex, the computational demands for training increase. The challenge lies in developing algorithms that can learn efficiently both in terms of computational resources and the number of samples required.

2.2 Proximal Policy Optimization

In this section, we present the cutting-edge DRL algorithm, Proximal Policy Optimization (PPO) (Schulman et al., 2017). PPO has demonstrated its capability in training successful agents for complex games as DotA 2 (Berner et al., 2019). Additionally, its effectiveness extends to real-world applications, ranging from dexterous in-hand manipulation by robotic hands (Andrychowicz et al., 2020) to addressing chip placement challenges in the semiconductor domain (Mirhoseini et al., 2020).

To establish PPO's background, we trace its lineage with its foundational predecessors REINFORCE (Williams, 1992) and Advantage Actor-Critic (A2C) (Mnih et al., 2016), which are based on the policy gradient. Building upon REINFORCE, several progressive enhancements lead to the realization of PPO.

The journey begins by transforming REINFORCE into an actor-critic algorithm through the incorporation of a learned value function, acting as a critic. This critic is further

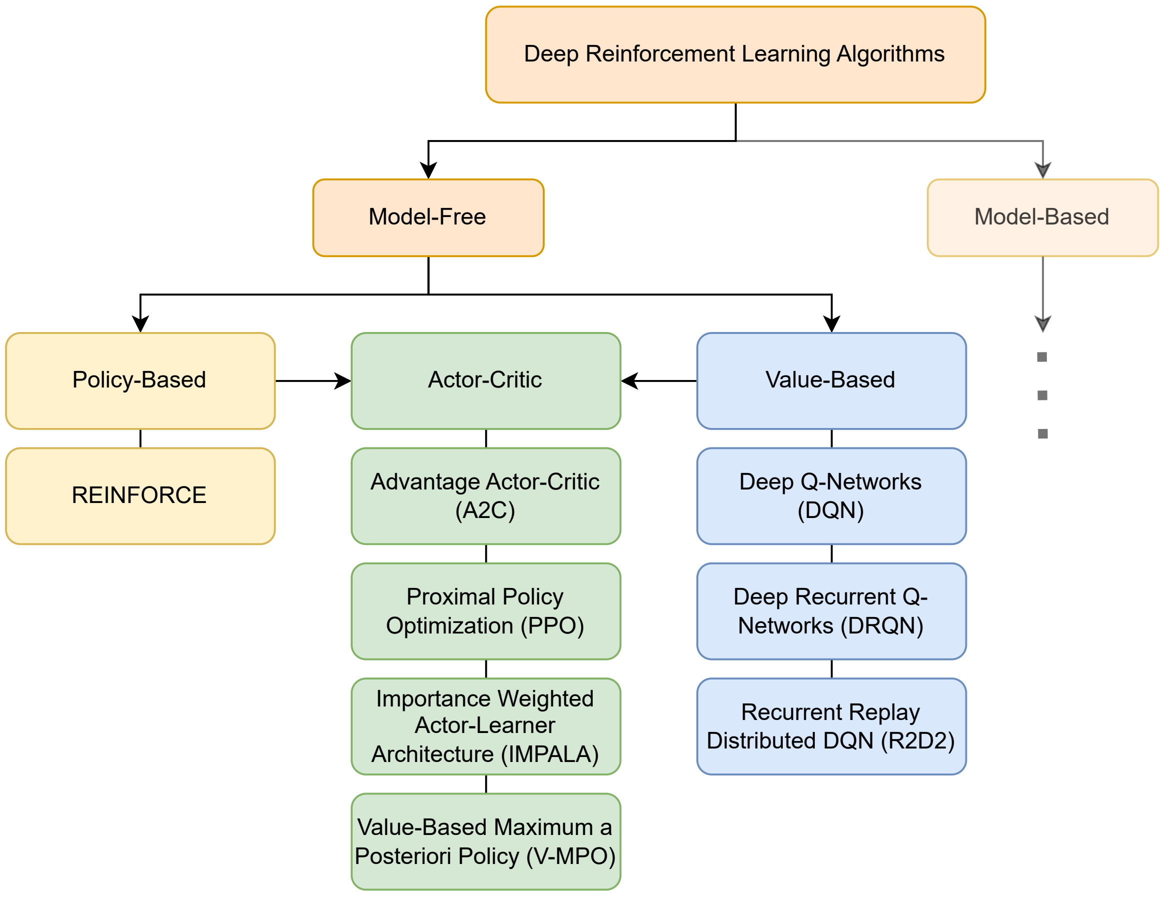

Deep Reinforcement Learning Algorithms
Model-Free
Model-Based
Policy-Based
Actor-Critic
Value-Based
REINFORCE
Advantage Actor-Critic (A2C)
Proximal Policy Optimization (PPO)
Importance Weighted Actor-Learner Architecture (IMPALA)
Value-Based Maximum a Posteriori Policy (V-MPO)
Deep Q-Networks (DQN)
Deep Recurrent Q-Networks (DRQN)
Recurrent Replay Distributed DQN (R2D2)

Figure 2.3: Taxonomy of Deep Reinforcement Learning Algorithms touched in this work. This study focuses only on model-free algorithms. The policy gradient algorithm REINFORCE is the foundation to A2C, which introduces the concept of actor-critic algorithms. Its successor, PPO, is pivotal for our baselines. Other algorithms are referenced due to their significance in related work. Actor-critic combines both value-based and policy-based approaches.

refined by adopting Generalized Advantage Estimation (GAE) (Schulman et al., 2016), paving the way for the introduction of PPO's distinct clipped surrogate objective. To encourage exploration, an entropy bonus augments this objective. Before delving into the final PPO algorithm, we detail the to-be-minimized loss functions and the approach for collecting training data concurrently.

Throughout related work (Chapter 3) and our exploration leading to PPO, we also touch on further algorithms, with Figure 2.3 visualizing a brief taxonomy.

2.2.1 REINFORCE

REINFORCE (Williams, 1992) is one of the foundational algorithms in the realm of policy gradient methods in RL. Unlike value-based approaches which seek to learn the value of being in a given state or taking a certain action, policy gradient methods, like REINFORCE, aim directly at optimizing the policy itself. Specifically, REINFORCE works by adjusting the policy parameters in a direction that maximizes expected returns, relying on the gradient of the expected return with respect to the policy parameters.

To implement REINFORCE, three major components are required:

- A policy defined by parameters, often represented by an artificial neural network.
- A well-defined objective function.
- A technique to optimize the policy parameters, typically relying on gradient ascent.

At its heart, the policy, denoted as π, is parameterized by θ — parameters of a neural network — that maps a state to its corresponding action probabilities. Actions can be determined deterministically by choosing the action with the highest probability or stochastically by sampling from the action probability distribution. The later is more prevalent as it aids exploration.

The objective function strives to optimize the expected discounted cumulative rewards (i.e. return):

$$J(\theta) = \mathbb{E}_{\tau \sim \pi}\left[\sum_{t=0}^{T} G_t(\tau)\right] \tag{2.21}$$

where τ symbolizes a trajectory sampled using the policy π_θ. The gradient of this objective, known as the policy gradient, is given by:

$$\nabla J(\theta) = \mathbb{E}_{\tau \sim \pi}\left[\sum_{t=0}^{T} G_t(\tau) \nabla \, log \, \pi(a_t|s_t, \theta)\right] \tag{2.22}$$

Here, $\nabla_\theta \, log \, \pi_\theta(a_t|s_t)$ constitutes the gradient of the log-probability of selecting action a_t in state s_t governed by the policy π_θ. For a complete derivation of the policy gradient, refer to (Graesser and Keng, 2020, p. 28).

Algorithm 1 REINFORCE

```
Require: Policy parameters $\theta$, learning rate $\alpha$
1: for episode $0, \dots, NumEpisodes$ do
2:     Sample trajectory $\tau$ using policy $\pi_\theta$
3:     for each step $(s_t, a_t, r_{t+1})$ in reverse order of $\tau$ do
4:         Compute return $G_t(\tau)$ starting from $t$
5:         $\theta \leftarrow \theta + \alpha \cdot G_t(\tau) \nabla \, log \, \pi(a_t|s_t, \theta)$
6:     end for
7: end for
```

The last component is gradient ascent to optimize θ:

$$\theta \leftarrow \theta + \alpha \nabla J(\theta) \tag{2.23}$$

The learning process is based on MC prediction (Section 2.1.7) and is depicted by Algorithm 1.

Since REINFORCE is an on-policy method, it does not reuse previously acquired training data, leading to potential sample inefficiencies when the agent-environment interactions are costly. Another notable challenge with this algorithm is its suffering from high variance. Incorporating off-policy data to enhance sample efficiency might amplify this variance, potentially leading to catastrophic changes in the policy gradient. Even minor changes in the policy gradient can drastically alter the policy. In cases where the policy collapses, it might not recover, given that this very policy is responsible for collecting the next trajectory. Such a phenomenon is termed "performance collapse".

REINFORCE is the foundation to PPO, but it necessitates various improvements to fulfill PPO. The following section explores the notion of subtracting a baseline aiming at stabilizing training.

2.2.2 REINFORCE with Baseline

REINFORCE, though foundational, benefits from several refinements, one of which is the incorporation of a baseline $b_{line}(s_t)$. By subtracting this baseline from the return G_t, a point of reference is established, effectively indicating whether the realized return was above or below expectations.

$$\nabla J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \big(G_t(\tau) - b_{line}(s_t) \big) \nabla \, log \, \pi(a_t|s_t, \theta) \right] \tag{2.24}$$

While the baseline could simply be the mean return over a trajectory:

$$b_{line} = \frac{1}{T} \sum_{t=0}^{T} G_t(\tau) \tag{2.25}$$

a more advanced strategy employs a learned value function $V(s_t, \theta)$ parameterized by θ:

$$\nabla J(\theta) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{T} \big(G_t(\tau) - V(s_t, \theta) \nabla\, log\, \pi(a_t | s_t, \theta) \big) \right] \tag{2.26}$$

Training the value function can parallel the policy's training mechanism in MC style. The value function and the policy can also rely on separate parameters, if desired. However in this work, shared parameters are the default.

Instead of an objective function, the emphasis shifts to minimizing the Mean Squared Error (MSE) (James et al., 2021, p. 29):

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \tag{2.27}$$

$\hat{y}_i$ is the predicted value, y_i is the target value, and n is the number of data points. In the context of REINFORCE, $\hat{y}_i$ is represented by $V(s_t, \theta)$ and y_i corresponds to $G_t(\tau)$ for each time step t. The final loss to minimize is:

$$L^V(\theta) = \frac{1}{T} \sum_{t=0}^{T} (V(s_t, \theta) - G_t(\tau))^2 \tag{2.28}$$

As the policy gradient and a learned value function are utilized in REINFORCE with baseline, one could assume that this concept refers to the idea of actor-critic methods as PPO. However, this is not the case yet, as elaborated next.

2.2.3 Actor-Critic

Actor-critic algorithms fuse two key components: the actor and the critic. The actor is responsible for determining the action to take in a given state, effectively directing how the agent should behave. On the flip side, the critic examines the action chosen by the actor, supplying feedback through a value estimate. This feedback refines the updates made to the actor, enhancing its decision-making process. To emphasize, the return used to reinforce the actor is completely replaced by the critic's feedback.

REINFORCE combined with a learned value function as a baseline (Equation 2.26) does not strictly qualify as an actor-critic method. The distinction lies in how states and actions are assessed. In this setup, the learned value function evaluates only the initiating state of a transition. The outcome of the action taken, is determined by the sampled return. However, for a genuine actor-critic approach, the subsequent state resulting from the action should also be evaluated by the value function.

One prevalent approach in actor-critic methods to learning the value function is through TD learning (Section 2.1.8), as seen in works like (Mnih et al., 2016; Schulman et al., 2017; Sutton and Barto, 2018). For simplicity, we transform REINFORCE with baseline to an actor-critic method by leveraging TD(0). The return in Equation 2.26 can be reinterpreted as the one-step return $G_{t:t+1}$. Thus, the return in REINFORCE with baseline's objective can be replaced with the state's look-ahead value estimation:

$$J(\theta) = \mathbb{E}_t \left[\left(R_{t+1} + \gamma V(s_{t+1}, \theta) - V(s_t, \theta) \right) log\, \pi(a_t|s_t, \theta) \right] \tag{2.29}$$

Here, the expression $R_{t+1} + \gamma V(s_{t+1}, \theta) - V(s_t, \theta)$ approximates the advantage function, introduced in Equation 2.14. Employing an n-step return gives rise to the advantage function $\hat{A}_{\pi}^{nstep}$ as applied in A2C's objective (Mnih et al., 2016):

$$\hat{A}_{\pi}^{nstep}(s_t, a_t) = \sum_{i=0}^{n-1} \left(\gamma^i R_{t+i} + \gamma^n V(s_{t+n}, \theta) - V(s_t, \theta) \right) \tag{2.30}$$

$$J(\theta) = \mathbb{E}_t \left[log\, \pi(a_t|s_t, \theta)\, \hat{A}_{\pi}^{nstep}(s_t, a_t) \right] \tag{2.31}$$

It is essential to note that we employ the parameters θ when formalizing A2C's advantage function. Although the policy and value function may share parameter sets, decoupling parameters between these components is also a viable strategy.

As the critic is known for stabilizing the training of policy gradient algorithms, PPO yields further improvements by leveraging GAE (Schulman et al., 2016), introduced next.

2.2.4 Generalized Advantage Estimation

Generalized Advantage Estimation (GAE) (Schulman et al., 2016) offers an advanced alternative to n-step returns (Section 2.1.9). It brings in a new parameter, $\lambda \in [0, 1]$, that elegantly smooths between the TD error and the actual sampled return. Unlike n-steps, which demand a specific choice of n, GAE provides a continuous balance.

Drawing inspiration from Graesser and Keng (2020), the breakdown of GAE can be presented in smaller steps. We start with three instances of the n-steps advantage function. To later simplify the notation, the variable δ_t is defined:

$$\begin{aligned}
\hat{A}_{t:t+1} &\doteq \delta_t && = R_t + \gamma V(s_{t+1}) - V(s_t) \\
\hat{A}_{t:t+2} &\doteq \delta_t + \gamma\delta_{t+1} && = R_t + \gamma R_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t) \\
\hat{A}_{t:t+3} &\doteq \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} && = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 V(s_{t+3}) - V(s_t)
\end{aligned} \tag{2.32}$$

With this foundation, GAE constructs an exponentially-weighted average of the n-step advantage estimators:

$$\hat{A}_{\pi}^{GAE}(s_t, a_t) = (1 - \lambda)(\hat{A}_{t:t+1} + \lambda\hat{A}_{t:t+2} + \lambda^2\hat{A}_{t:t+3} + \ldots) \tag{2.33}$$

This can be compactly represented as (Graesser and Keng, 2020, p. 139-140):

$$\hat{A}_{\pi}^{GAE} = \sum_{l=0}^{\infty} \left((\gamma\lambda)^l \delta_{t+l} \right) \tag{2.34}$$

In essence, GAE offers a flexible approach to smoothly balance variance and bias. In the following section, we integrate GAE as the critic in the forthcoming objective of PPO.

2.2.5 Clipped Surrogate Objective

The motivation behind PPO is to prevent overly aggressive updates that can destabilize learning, leading to performance collapse in the worst case. Schulman et al. (2017) introduce a surrogate objective that undergoes clipping to ensure that the policy update remains within a trust region, effectively moderating the extent of each update and preventing excessive deviations from the current policy. This approach strikes a balance, allowing for meaningful policy improvements while mitigating the risk of harmful large policy shifts. The following description of PPO's objective formulation leans on the notation presented by Graesser and Keng (2020).

The foundation for PPO is the policy gradient objective of A2C (Equation 2.31) combined with GAE:

$$J(\theta) = \mathbb{E}_t \left[log\, \pi(a_t|s_t, \theta)\, \hat{A}_\pi^{\mathrm{GAE}}(s_t, a_t) \right] \tag{2.35}$$

Transitioning to the surrogate objective is facilitated by the method of importance sampling (Schulman et al., 2015). An essential construct is the probability ratio, denoted as $ratio_t(\theta)$:

$$ratio_t(\theta) = \frac{\pi(a_t|s_t, \theta)}{\pi(a_t|s_t, \theta_{\mathrm{old}})} \tag{2.36}$$

Here, θ_{old} refers to the policy parameters under which the trajectory data was sampled. This ratio captures the change in policy for action a_t in state s_t from the old policy to the current one. By incorporating this ratio, the surrogate objective, called Conservative Policy Iteration (CPI) (Kakade and Langford, 2002), is established:

$$J^{\mathrm{CPI}}(\theta) = \mathbb{E}_t \left[ratio_t(\theta)\, \hat{A}_\pi^{\mathrm{GAE}}(s_t, a_t) \right] \tag{2.37}$$

To further mitigate large policy updates, PPO incorporates a clipped version of this surrogate objective:

$$J^{\mathrm{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(ratio_t(\theta) \hat{A}_\pi^{\mathrm{GAE}}(s_t, a_t),\, \mathrm{clip}(ratio_t(\theta), 1-\epsilon, 1+\epsilon) \hat{A}_\pi^{\mathrm{GAE}}(s_t, a_t) \right) \right] \tag{2.38}$$

Given the clip range hyperparameter ϵ, the clip function constrains $ratio_t(\theta)$ to the clipping neighborhood $|ratio_t(\theta) - 1| \leq \epsilon$. Choosing smaller values for ϵ results in more conservative policy updates. When $ratio_t(\theta)$ lies within the range $[1-\epsilon, 1+\epsilon]$, both expressions within the min() function are equivalent. Otherwise, the min function selects the smaller value, effectively providing a pessimistic bound on the unclipped objective.

The significance of centering this clipping region around the value of 1 becomes evident when understanding the role of $ratio_t(\theta)$. When the new policy mirrors the old one, the equation becomes:

$$ratio_t(\theta) = ratio_t(\theta_{\text{old}}) = \frac{\pi(a_t|s_t, \theta_{\text{old}})}{\pi(a_t|s_t, \theta_{\text{old}})} = 1 \tag{2.39}$$

This indicates a perfect overlap between the two policies. Conversely, as the new policy diverges from the old, the value of $ratio_t(\theta)$ shifts away from 1, encapsulating both the magnitude and direction of this divergence. When multiplied by the advantage estimate, this deviation takes on heightened importance, enhancing the utility of the clipping mechanism.

With the clipped surrogate objective now established for policy updates, the only remaining element to introduce is the entropy bonus, as discussed subsequently.

2.2.6 Entropy Bonus

Overly deterministic policies can quickly converge to sub-optimal solutions if exploration is not encouraged. This is where the concept of an entropy bonus comes into play. Entropy, in the context of RL, measures the randomness or unpredictability of the actions chosen by a policy. A higher entropy suggests a more exploratory policy, while lower entropy indicates a more deterministic one.

Mathematically grounded in the concept of the Shannon entropy (Shannon, 1948), the entropy H of a policy π for a state s is represented as:

$$H(\pi(s)) = -\sum_{a\in\mathcal{A}} \pi(a|s)\, log\, \pi(a|s) \tag{2.40}$$

Where s is a given state, and a ranges over all possible actions. This bonus is essentially a term added to PPO's loss, which is proportional to the entropy of the policy. By doing so, the algorithm is incentivized to maintain a degree of stochasticity in its policy.

With the entropy bonus in place, the final loss function of PPO is composed next.

2.2.7 Loss Function

In optimizing both the policy and the value function within the PPO algorithm, it is essential to derive the loss functions that guide the learning. This section illustrates the formulation of these loss functions and how they integrate to form the complete loss for PPO.

The clipped surrogate objective $J_t^{\text{CLIP}}(\theta)$ is transformed into a loss function, $L_t^C(\theta)$, by taking its negative:

$$L_t^C(\theta) = -\mathbb{E}_t \left[\min\left(ratio_t(\theta)\hat{A}_\pi^{\text{GAE}}(s_t, a_t),\ \text{clip}\left(ratio_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_\pi^{\text{GAE}}(s_t, a_t)\right)\right] \tag{2.41}$$

The loss associated with the value function is conventionally defined using the squared error between the predicted value and the target value:

$$\left(V(s_t,\theta) - V_t^{\text{tar}}\right)^2 \tag{2.42}$$

Nevertheless, in line with the knowledge from the code implementation by Schulman et al. (2017) and as highlighted by Engstrom et al. (2020), the value function can also be clipped as the policy objective. We adopt this clipped version as well:

$$L_t^V(\theta) = \max\left[\left(V(s_t,\theta) - V^{\text{tar}}\right)^2, \left(\text{clip}\left(V(s_t,\theta), V(s_t,\theta_{\text{old}}) - \epsilon, V(s_t,\theta_{\text{old}}) + \epsilon\right) - V^{\text{tar}}\right)^2\right] \tag{2.43}$$

To conclude, the comprehensive loss function combines the policy loss, the value function loss, and the entropy bonus to encourage exploration:

$$L_t^{C+V+H}(\theta) = \mathbb{E}_t\left[L_t^C(\theta) + c_1 L_t^V(\theta) - c_2 H_t[\pi_\theta](s_t)\right] \tag{2.44}$$

In this final form, c_1 and c_2 are coefficients that control the balance between policy optimization, value function fitting, and entropy-based exploration. In the subsequent section, we will explore the methodology for collecting training data, setting the final stage for a comprehensive presentation of the entire algorithm.

2.2.8 Data Parallelism

Before presenting the entire PPO algorithm, it is worth noting the commonly used approaches to gathering and utilizing training data for optimization. PPO follows the notion of A2C, which is a variant based on Asynchronus Advantage Actor-Critic (A3C) (Mnih et al., 2016). Both these algorithms are built upon REINFORCE with baseline (Section 2.2.2), where the value function is estimated using n-steps. Their distinctions emerge in how they leverage data parallelism.

The asynchronicity of A3C arises from the deployment of multiple actor-learner pairs that operate independently across different threads. Within this setup, every actor-learner initiates by synchronizing its parameters with the global ones intended for optimization. Then, a trajectory of fixed length is gathered to update the global parameters.

A2C deviates from this approach in its handling of parallelism. Instead of pairing both the actor and learner in tandem, A2C allocates one single actor to each thread. Collectively, these actors collect numerous trajectories, which are subsequently utilized by a single learner for parameter updates. Post optimization, the actors recommence data sampling, now informed by the refined policy.

One might question the motivation behind deploying multiple actors. The advantages are twofold: Parallel data collection enhances efficiency, and perhaps more importantly, it aids in data decorrelation, an attribute observed to be pivotal in the experience replay mechanism as manifested in Deep Q-Networks (DQN) (Mnih et al., 2015). Through the engagement of multiple actors, varied instances of the environment are explored, leading to encounters with a diverse set of states.

According to Schulman et al. (2017), A2C performed as well as or better than A3C.

2.2.9 Algorithmic Details

Thus far, we have dissected the various components essential to PPO's architecture:

- **REINFORCE:** The foundation of policy gradient algorithms that provide a direct mapping from states to actions.
- **Actor-Critic:** A synergy where the actor's optimization is informed by feedback from the critic.
- **Generalized Advantage Estimation:** A balanced approach merging the strengths of Monte Carlo prediction and temporal difference learning to refine the value function.
- **Clipped Surrogate Objective:** This concept ensures policy updates are both constrained to prevent destabilization and significant enough to drive meaningful learning.
- **Entropy Bonus:** A mechanism to bolster exploration during the learning process.
- **Final Loss Functions:** The target expressions to be minimized in the optimization process.
- **Data Sampling:** Reflecting techniques to gather training data akin to A2C.

The pseudocode for understanding PPO's mechanics is presented in Algorithm 2. A typical PPO iteration begins by sampling training data concurrently through multiple actors over a predefined number of steps. These collected trajectories could span multiple episodes or truncated segments of episodes, which will continue in the subsequent iteration. Such continuity is feasible because PPO does not rely solely on MC estimates. After data collection, the GAE is computed. This data is then shuffled into random mini-batches, serving multiple epochs of optimization, with the objective to minimize the loss as expressed in Equation 2.44. Notably, after processing the first mini-batch, the policy changes. This means subsequent mini-batches and epochs deal, to some extent, with off-policy data. In contrast, A2C runs a single epoch over the complete batch. The resilience of PPO in managing data derived from a marginally outdated policy, potentially due to its clipped surrogate objective, makes it more sample-efficient compared to its predecessors (Schulman et al., 2017).

While PPO is a popular and powerful algorithm, its inner workings are still somewhat mysterious. One reason for this can be found in the gaps that are apparent in the original publication (Schulman et al., 2017). For instance, advantages can be normalized prior optimization or the norm of the gradients can be clipped. Many researchers have taken on the challenge of demystifying PPO's details, with studies such as those by Engstrom et al. (2020); Andrychowicz et al. (2021); Huang et al. (2022a).

Algorithm 2 Proximal Policy Optimization (PPO)

```
Require: Parameters $\theta$, learning rate $\alpha$, clip epsilon $\epsilon$, loss coefficients $c_1$ and $c_2$
Require: Discount factor $\gamma$, smoothing parameter $\lambda$, mini-batch size $M$
 1: for iteration $= 0, 1, \ldots,$ Iterations do
 2:     // Sample trajectories using $\pi_\theta$
 3:     for each actor do
 4:         for step $= 0, 1, \ldots,$ Steps do
 5:             Sample agent-environment interaction transition using $\pi_\theta$
 6:         end for
 7:     end for
 8:
 9:     // Generalized advantage estimation
10:     for each trajectory $\tau_i$ do
11:         for each step $(s_t, a_t, r_{t+1}, V_t)$ in reverse order of $\tau_i$ do
12:             Compute advantages $\hat{A}_t^{GAE}$
13:         end for
14:     end for
15:
16:     // Optimize parameters by minimizing $L^{C+V+H}$
17:     for epoch $= 0, 1, \ldots, E$ do
18:         for mini-batch of $M$ samples do
19:             // Compute, aggregate, and combine losses
20:             $L^C(\theta) = -\mathbb{E}\left[\sum_{t=0}^{T} \min\left(ratio_t(\theta)\hat{A}_\pi^{GAE}, \text{clip}(ratio_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_\pi^{GAE}\right)\right]$
21:
22:             $L^V(\theta) = \frac{1}{T}\sum_{t=0}^{T}\left(V(s_t, \theta) - V_t^{\text{target}}\right)^2$
23:
24:             $L^{C+V+H}(\theta) = L^C(\theta) + c_1 L^V(\theta) - c_2 \sum_{t=0}^{T} H[\pi_\theta](s_t)$
25:
26:             // Update policy parameters using gradient descent
27:             $\theta \leftarrow \theta - \alpha \nabla L^{C+V+H}(\theta)$
28:         end for
29:     end for
30: end for
```

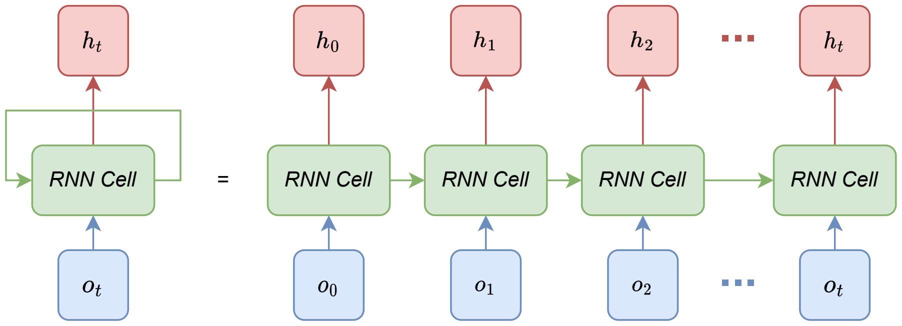
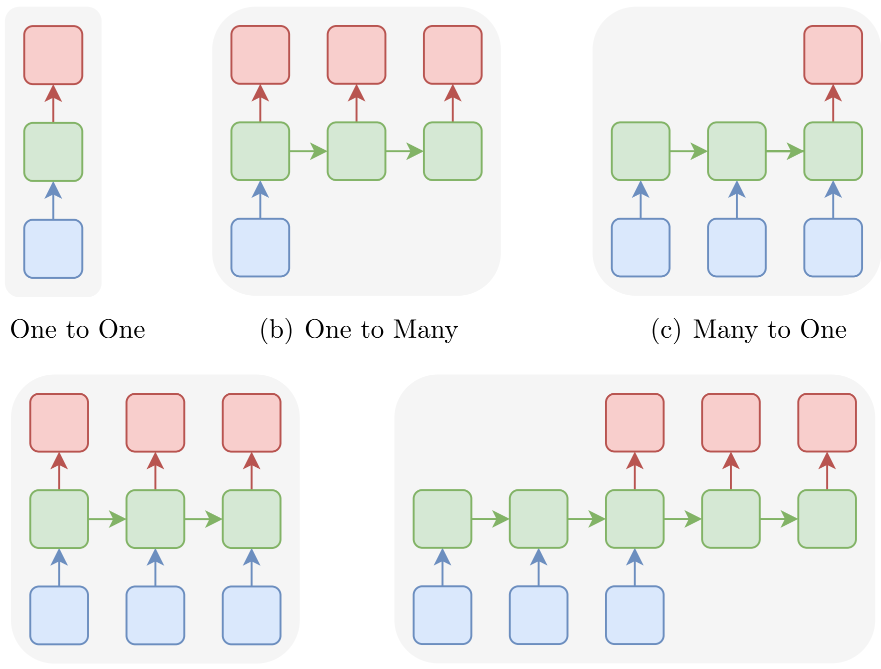


Figure 2.4: An unfolded representation of a Recurrent Neural Network processing a sequence. In DRL, it takes an observation, processes it, and produces a hidden state. Using an auto-regressive approach, subsequent steps incorporate both the previous hidden state and the new observation, generating a new hidden state. The parameters of the RNN are optimized using back-propagation through time (Werbos, 1990).

2.3 Recurrent Neural Networks

Deep Reinforcement Learning agents may be confronted with dynamic environments where traditional Artificial Neural Networks (ANNs) may be insufficient. Often, agents find themselves in scenarios where they must recall past interactions or adapt to sequences of observations that do not maintain a fixed size. Partial observations, a frequent occurrence in many real-time strategy games due to constraints like the fog of war, limit the agent's access to the complete game state. In such situations, only a limited "camera view" reveals the game's current state, challenging the agent to make decisions based on this truncated information. Furthermore, consider games like top-down shooters, where the number of enemy units can vary dynamically. Representing such fluctuating game states using a vector of fixed length becomes problematic. While zero padding the game state vector to a maximum fixed length is an option, it might not always be the most efficient or practical approach.

In situations as described above, relying solely on traditional Multi-Layer Perceptrons (MLPs) or Convolutional Neural Networks (CNNs) may fall short. This is where the flexibility and memory capabilities of Recurrent Neural Networks (RNNs) (Rumelhart et al., 1986) shine, offering a more adaptable solution for these DRL challenges. In the following sections, while providing more context on sequential tasks, we explore the original kind of RNNs that are known to suffer from exploding or vanishing gradients. Afterwards, we delve into the more sophisticated architectures of the Long Short-Term Memory (LSTM) and the Gated Recurrent Unit (GRU), which are both enhanced by leveraging gating mechanisms.



(a) One to One (b) One to Many (c) Many to One

(d) Many to Many (e) Many to Many (dynamic)

Figure 2.5: Different types of sequential task mappings. The one to one mapping represents a non-sequential task. The one to many mapping takes a single input and produces multiple outputs. In the context of DRL, the many to one mapping is crucial, where a history of observations leads to a singular current output, directing an action. There are also two many to many mappings: one with consistent input and output lengths, and another in a dynamic setting where not every step has an associated input or output. We typically use the term "sequence-to-one" to describe a many to one mapping.

2.3.1 Vanilla Recurrent Neural Networks

The roots of Recurrent Neural Networks (RNNs) (Rumelhart et al., 1986) can be traced back to several related architectures, each bearing similarities to notable models as the Hopfield network (Hopfield, 1982), the Elman network (Elman, 1990), and the Jordan network (Jordan, 1997). The term "vanilla RNN" typically refers to the foundational concept of Recurrent Neural Networks (RNNs), which are adept at handling sequential data. By iterating over sequence elements, these networks preserve a "memory" in the form of a hidden state. This mechanism ensures that previous sequence elements are retained and recalled, enabling the network to capture temporal dependencies and patterns. As a result, RNNs are applicable in applications like time-series forecasting, natural language processing, and more.

To conceptualize the operation of an RNN, imagine its layers being "unrolled" as a chain of duplicates across time or sequence steps. As illustrated in Figure 2.4, the recurrent layer, often referred to as a "cell", receives an observation o_t. This observation undergoes processing within the cell, outputting a hidden state that aims to provide a useful summary of past observations. In the subsequent step, the same cell, as parameters are the same, simultaneously accepts two inputs: the previously output hidden state and the upcoming observation o_{t+1}. This iterative procedure continues, processing the sequence piece by piece. These networks are characterized by their auto-regressive nature, where predictions for the next step are conditioned on previous steps. Moreover, the connections between hidden states are called recurrent connections.

Sequential tasks can be categorized based on their mappings: one input to many outputs, many inputs to many outputs, or many inputs to a single output, as depicted in Figure 2.5. Within DRL, the emphasis is on the last scenario, where only a present action is chosen based on a history of observations. Obviously, the agent does not make or reconsider decisions for the past.

Training RNNs involve an algorithm called Back-Propagation Through Time (Back-Propagation Through Time (BPTT)), which extends the concept of back-propagation used in traditional feed-forward networks (Werbos, 1990). In BPTT, the gradients are calculated by unrolling the entire sequence, meaning the error is back-propagated from the last time step all the way back to the first. This allows the network to update weights based on the entire sequence, ideally capturing long-range dependencies in the propagated data. However, back-propagating through these sequential steps leads to two notorious problems: vanishing and exploding gradients.

Concerning the vanishing gradient problem, gradients with magnitudes smaller than one, when propagated through multiple layers, shrink exponentially. This causes the weights in the initial layers (corresponding to earlier time steps) to receive minimal to no updates. Consequently, the RNN finds it challenging to learn and retain information from earlier sequence steps, rendering them ineffective in capturing long-range dependencies.

Conversely, when gradients possess values greater than one, their magnitudes can amplify exponentially during back-propagation, resulting in them "exploding". Such behavior can induce extremely large weight updates, driving the model to oscillate wildly and hindering its ability to converge to a meaningful solution. A more detailed view on these gradient issues can be retrieved from (Goodfellow et al., 2016, pp. 396-399) and (Trask, 2019, pp. 272-273).

To alleviate the computational effort and gradient issues, truncated BPTT can be implemented. Rather than back-propagating through the entire sequence, this approach back-propagates only through a fixed number of steps, effectively segmenting the long sequence into smaller, more manageable chunks. Crucially, even though back-propagation is limited in these chunks, the initial hidden state for each new segment should not be reset to a default state, such as zeros. Instead, it should continue or "warm start" from the last hidden state of the previous segment. This ensures continuity in the sequence processing. As a result, while the sequence input to the network remains untruncated, the gradients computed during backpropagation are effectively limited to recent time steps. This adaptation is not without its trade-offs (Trask, 2019, p. 268). First, it introduces a slight bias; second, it reduces the span over which an RNN can directly learn to recall

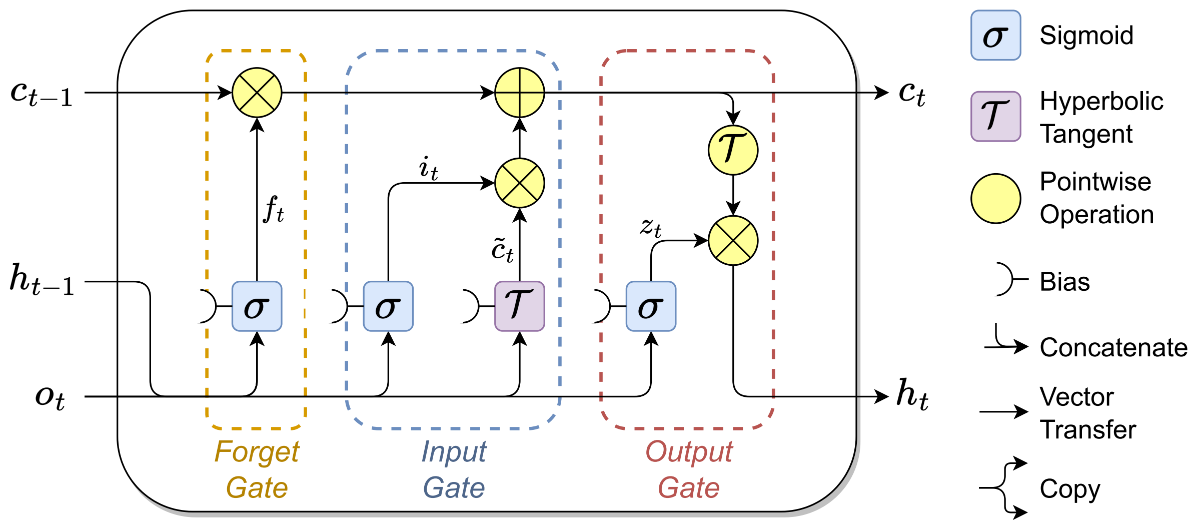


Figure 2.6: Schematic representation of a Long Short-Term Memory cell showcasing its three gates and interconnections with its cell state. The rounded rectangles represent fully connected layers that are activated by either sigmoid or the hyperbolic tangent function. Adapted from (Yu et al., 2019).

events. Due to the truncation of gradients in BPTT, the gradients cannot instruct the model to retain information beyond this point. Lastly, determining the truncation point is a hyperparameter that requires tuning.

In the upcoming subsections, we will explore the intricacies of advanced RNN architectures that employ gating mechanisms to address gradient-related and long-term dependency challenges.

2.3.2 Long Short-Term Memory

In addressing the limitations of vanilla RNNs, Hochreiter and Schmidhuber (1997) introduced the Long Short-Term Memory (LSTM). This innovation significantly enhanced the ability to retain information over prolonged sequences (Yu et al., 2019). The architecture of this recurrent mechanism has been further refined over time, and in our work, we adopt the version detailed by Graves (2012).

Central to the LSTM cell are its three gating mechanisms: the forget, input, and output gates (as shown in Figure 2.6). Each gate within the LSTM is typically modelled using an MLP (i.e. fully connected layer). This layer is then activated using the sigmoid function, which constrains its output to a range between 0 and 1. A value closer to 0 indicates a nearly closed gate, while one approaching 1 signifies an open gate. These gates collaboratively determine the flow of information — what to discard, retain, or transmit. Complementing this, the LSTM cell also incorporates a "cell state", serving as its intrinsic memory repository.

Unraveling the mechanics of an LSTM cell's forward pass, we begin with the forget gate, which was contributed by Gers et al. (2002). This gate evaluates the relevance of each piece of information in the cell state. By using the sigmoid function, the forget gate outputs values between 0 and 1, effectively deciding on which information to forget or to retain. The equation driving this gate is:

$$f_t = \sigma(W_{fh}h_{t-1} + W_{fo}o_t + b_f) \tag{2.45}$$

Here, the terms W and b denote the weight matrices and biases, o_t symbolizes the current time step's input observation, and h_{t-1} captures the prior hidden state.

Progressing further, the input gate and the cell state update mechanism come into play. The input gate is in charge of updating the cell state with new information. The candidate values for this update are represented by $\tilde{c}_t$. Both the decision and the potential update are expressed by:

$$i_t = \sigma(W_{ih}h_{t-1} + W_{io}o_t + b_i) \tag{2.46}$$

$$\tilde{c}_t = \tanh(W_{\tilde{c}h}h_{t-1} + W_{\tilde{c}o}o_t + b_{\tilde{c}}) \tag{2.47}$$

With the forget gate's and input gate's decisions at its disposal, the cell state c_t gets updated as:

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t \tag{2.48}$$

At last, the output gate z_t determines what portion of the cell state should be transmitted as the cell's output:

$$z_t = \sigma(W_{zh}h_{t-1} + W_{zo}o_t + b_z) \tag{2.49}$$

$$h_t = z_t \cdot \tanh(c_t) \tag{2.50}$$

As information flows through an LSTM cell, it undergoes a series of evaluations and transformations, driven by the three gates and the internal memory, ensuring that the model captures relevant temporal dependencies without being hampered by issues such as the vanishing or exploding gradient. Further insights, for example concerning the LSTM's backward pass, can be found in (Graves, 2012). A slightly more lightweight recurrent cell, the Gated Recurrent Unit, is introduced next.

2.3.3 Gated Recurrent Unit

The Gated Recurrent Unit (GRU), introduced by Cho et al. (2014), emerged as an alternative to the LSTM, aiming to simplify the recurrent unit's design while still efficiently capturing long-range dependencies. As portrayed in Figure 2.7, the GRU utilizes two gates: the update and reset gates. The primary distinction between a GRU and an LSTM is how it combines the hidden state and cell state into one entity. This combination makes

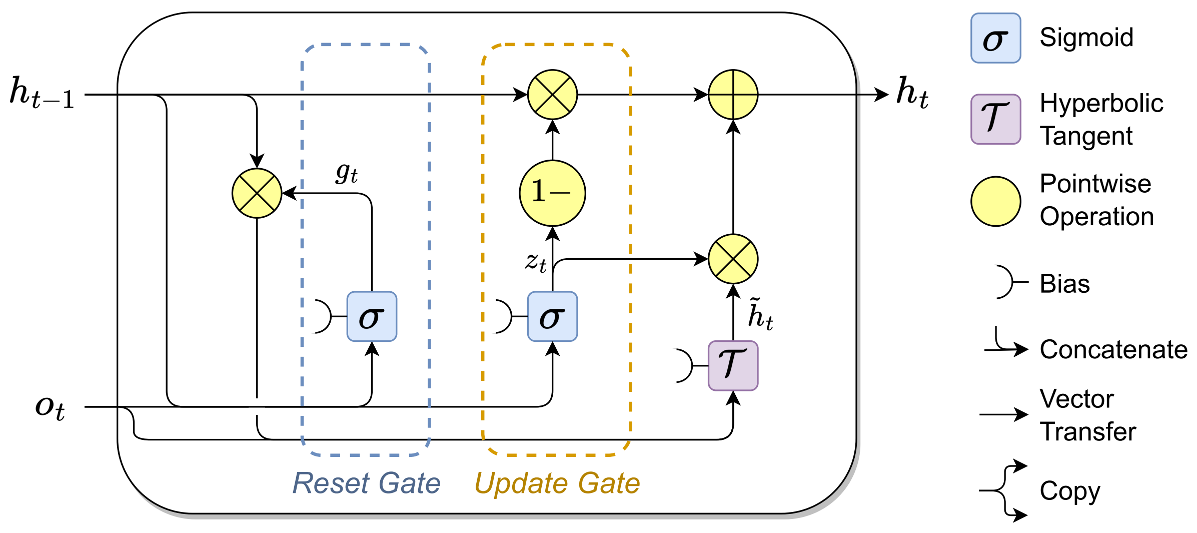


Figure 2.7: A depiction of the Gated Recurrent Unit cell illustrating its two gates and interconnections with the content propagated by the prior hidden state. The rounded rectangles represent fully connected layers that are activated by either sigmoid or the hyperbolic tangent function. Adapted from (Yu et al., 2019).

GRUs typically more computationally efficient and requires fewer parameters than LSTMs. However, this also implies a trade-off between model simplicity and capacity.

To understand the GRU cell's forward pass mechanism, the update gate, similar in function to the LSTM's forget and input gates, determines which information to discard and which new data to incorporate:

$$z_t = \sigma(W_{zh}h_{t-1} + W_{zo}o_t + b_z) \tag{2.51}$$

Meanwhile, the reset gate decides how much old information to forget, allowing the cell to ignore data that is no longer important:

$$g_t = \sigma(W_{gh}h_{t-1} + W_{go}o_t + b_g) \tag{2.52}$$

Subsequently, influenced by the reset gate's choice, a prospective hidden state undergoes computation:

$$\tilde{h}_t = \tanh(W_{\tilde{h}h}(g_t \cdot h_{t-1}) + W_{\tilde{h}o}o_t + b_{\tilde{h}}) \tag{2.53}$$

Lastly, using the update gate, a blend of the preceding hidden state and the candidate hidden state is created, producing this time step's final hidden state.

$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \tilde{h}_t \tag{2.54}$$

GRU offers a balance between computational efficiency and the ability to capture long-range dependencies. Selecting between LSTMs and GRUs is often problem-specific, and it may require empirical evaluation to ascertain the most suitable architecture for a given task. Research by, for instance, Chung et al. (2014) and Morad et al. (2023), demonstrates the competitiveness of GRU.

2.4 Transformer-XL

A strong alternative to RNNs is the Transformer architecture (Vaswani et al., 2017). The primary difference between the two is their approach to processing input sequences. Whereas RNNs propagate a sequence piece by piece, transformers process the entire sequence simultaneously, offering enhanced parallelization. This capability is largely attributed to employing an attention mechanism. Attention allows the model to focus on different parts of the input sequence, dynamically weighing the significance of each part in relation to others. This mechanism not only enables efficient parallel processing but also helps the model capture long-range dependencies and relationships within the data. Consequently, transformers have profoundly influenced the fields of language understanding and image processing (Tay et al., 2023) and have showcased superior performance in DRL (Parisotto et al., 2020; Lampinen et al., 2021).

Concerning sequence-to-sequence tasks, transformers employ both an encoder and a decoder. However, for the scope of this research, focused on DRL tasks, which are considered as sequence-to-one problems (Section 2.3.1), only the encoder's functionality is relevant. Subsequently, we detail the concept of attention, Multi-Head Attention (MHA), and positional encodings. Our exploration further shifts to two distinctive elements of the Transformer-XL (TrXL) that differentiate it from the original transformer. The foremost is a segment-level recurrence mechanism, enabling the transformer to engage with inputs beyond a predetermined sequence length. Given the inherent permutation invariance of attention and extended sequence lengths, the incorporation of a relative positional encoding becomes necessary. At last, we briefly detail the incorporation of GRU cells to advance TrXL to GTrXL.

2.4.1 Scaled Dot-Product Attention

The Scaled Dot-Product Attention (Vaswani et al., 2017) enables the transformer to selectively emphasize certain parts of the input data when formulating an output.

The mechanism involves three major matrices: Query (Q), Key (K), and Value (V). Drawing an analogy with relational databases helps make their roles more tangible:

- Query (Q): Analogous to a database query, it represents the current focus or the question being posed.

- Key (K): Resembling the index or metadata in databases, keys provide a "labeling system" for the input data, indicating how relevant each part is to a given query.

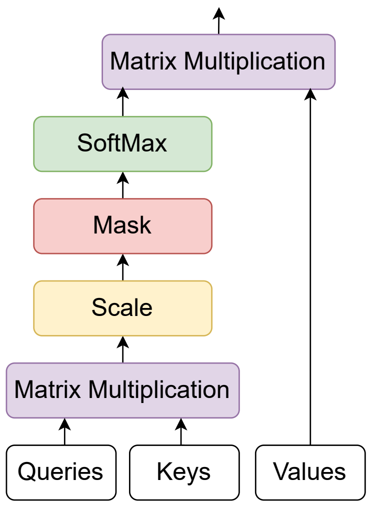


Figure 2.8: Scaled Dot Product Attention. Adapted from (Vaswani et al., 2017).

- Value (V): Much like the actual rows of data in a database table, values represent the content of the input data.

However, unlike a traditional database that outputs a discrete set of rows, the attention mechanism returns a compressed context vector by:

1. Computing the dot product of the query with every key, resulting in a set of scores. These scores represent the relation between the current focus and each label.
2. Scaling these scores by dividing by the square root of the dimension of the key vectors (denoted by d_k). This scaling step ensures stability in subsequent softmax computations (Vaswani et al., 2017).
3. Introducing a mask to the scaled scores. This mask selectively ensures that specific values are not considered during attention by assigning these values a large negative number, effectively pushing their softmax output towards zero.
4. Applying the softmax function to these scaled scores to derive a set of weights that collectively sum to 1.
5. Multiplying these weights with their respective values. This produces a weighted combination of the input data, emphasizing segments with higher attention scores.

The attention operation can be summarized by Figure 2.8 and this equation:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V \tag{2.55}$$

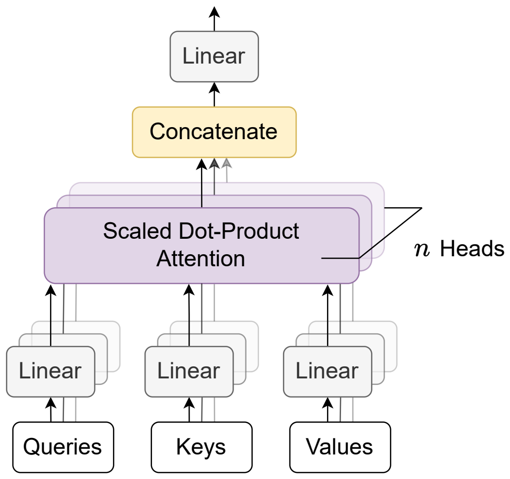


Figure 2.9: Multi-Head Attention. Adapted from (Vaswani et al., 2017).

2.4.2 Multi-Head Attention

Multi-Head Attention (MHA) augments the capabilities of the standard attention mechanism by enabling the model to concurrently focus on different parts of the input sequence from multiple perspectives or "heads" (Vaswani et al., 2017). By doing so, it enhances the model's ability to capture various types of relationships within the data.

In MHA, each head computes its own set of query, key, and value projections and then uses these projections to compute its attention scores. By processing the sequence in this manner, each head can potentially capture different patterns within the data. For example, while one head might focus on short-term dependencies, another might capture long-term relationships.

To implement this, the original query, key, and value matrices are independently transformed using n linear layers, corresponding to n heads. These linear transformations yield different representation subspaces for each head, allowing them to capture unique patterns.

Once all the heads compute their individual attention results, their outputs are concatenated. This concatenated output then undergoes another linear transformation, which is achieved using a fully connected layer, to produce the final output of the MHA.

The operation of MHA is visualized by Figure 2.9 and can be expressed as:

$$\begin{aligned} \mathrm{MHA}(Q,K,V) &= \mathrm{Concatenate}(\mathrm{head_1},\dots,\mathrm{head_n})W^O \\ \text{where } \mathrm{head_i} &= \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{2.56}$$

In this equation:

- $\mathrm{head_i}$ represents the output from the i^{th} attention head.

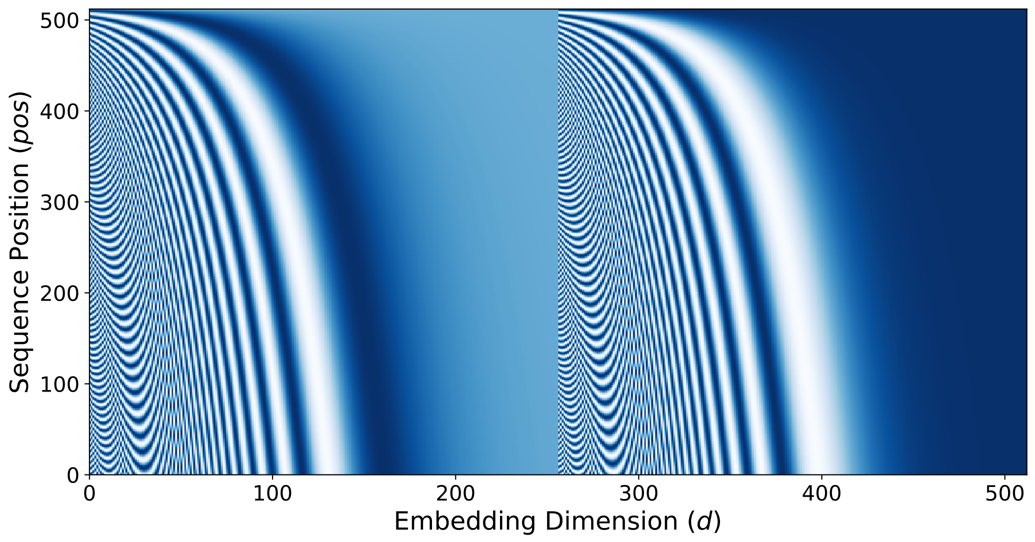


Figure 2.10: Sinusoidal Positional Encoding

- W_i^Q, W_i^K, and W_i^V are the linear projection matrices for the query, key, and value for the i^{th} head.
- W^O is the output linear projection matrix applied after concatenation of all head outputs.

2.4.3 Absolute and Learned Positional Encodings

The described attention mechanism does not have an inherent sense of order or position for the input sequence. This means that if the order of elements in the input sequence is permuted (i.e. rearranged), the attention mechanism by itself would treat them the same way. To address this limitation, the transformer architecture incorporates a positional encoding. This encoding embeds information about the position of each item within the sequence, ensuring that the model can account for the order of elements when processing them.

Vaswani et al. (2017) introduced a sinusoidal positional encoding scheme. This encoding is designed to be added to the embeddings of the sequence elements. The sinusoidal positional encoding, is visualized in Figure 2.10 and defined by the following equations:

$$P_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right) \tag{2.57}$$

$$P_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right) \tag{2.58}$$

Where:

- P represents the positional encoding matrix.
- pos is the position of an item within the sequence.
- i is the dimension of the encoding, ranging from 0 to $d-1$, where d is the depth of the embedding.
- The terms $2i$ and $2i+1$ ensure that the sine function is applied to even dimensions and the cosine function to odd dimensions.
- The division by $10000^{2i/d}$ results in a unique frequency for each dimension, allowing the model to distinguish positions across a wide range of sequence lengths.

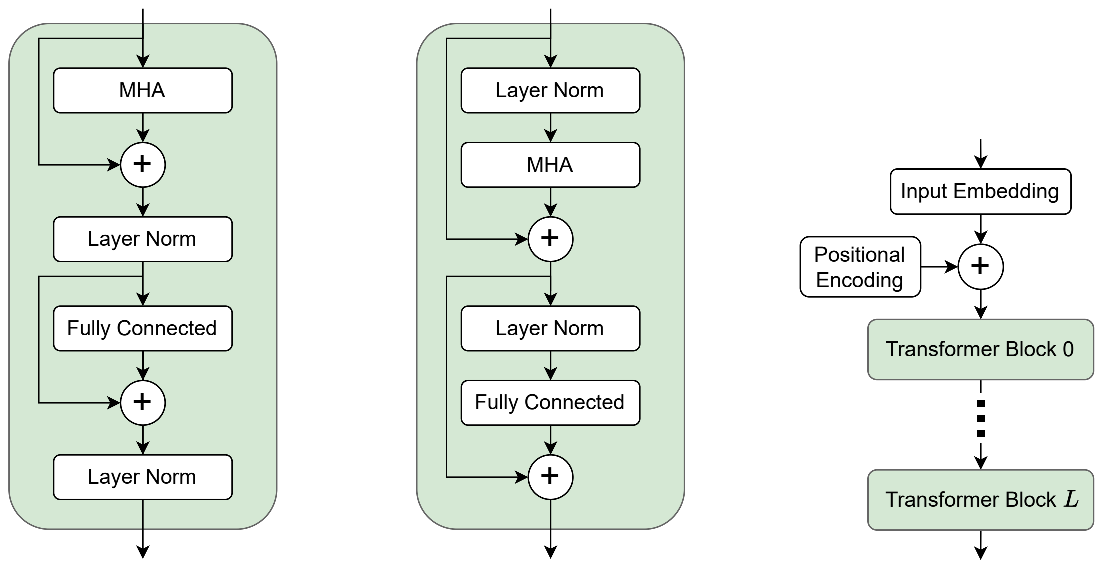
Apart from the sinusoidal positional encoding, there is also the concept of a learned positional encoding. In this approach, rather than using a fixed mathematical formula to determine the positional encodings, the model is allowed to learn them during training. This is achieved by initializing a matrix of positional encodings, which is then updated using backpropagation, similar to how other model parameters are learned. Vaswani et al. (2017) show that both approaches achieve equivalent results, but favor the sinusoidal nature of the encoding, because it ensures that the model can extrapolate to sequence lengths beyond those seen during training.

In practice, these positional encodings are added directly to the embeddings of the sequence items before they are fed into the transformer and therefore belong to the family of Additive Positional Encoding (APE) (Dufter et al., 2022). This addition ensures that the model has access to both the semantic content of each item and its position within the sequence, enabling it to process sequences with an understanding of order and context. One might wonder if the addition of positional encoding might overshadow or distort the original content of the sequence. However, a helpful analogy is to think of positional encoding as a subtle background tune in a song. While the tune is present and adds depth to the song, the main melody (content of the sequence) remains prominent and recognizable. Similarly, the positional encoding provides the model with a sense of order without overwhelming the primary content of the sequence.

2.4.4 The Transformer Encoder

A transformer block comprises the following components:

- Multi-Head Attention (MHA) (Section 2.4.2),
- Positional Encoding (Section 2.4.3),
- Fully Connected Layer (often termed as MLP or linear layer),
- Layer Normalization (or Layer Norm) (Ba et al., 2016), and



(a) Post-Layer Normalization (b) Pre-Layer Normalization (c) Transformer Encoder

Figure 2.11: Illustration of the transformer encoder, based on L stacked transformer blocks.

- Residual Connections (He et al., 2016).

Figures 2.11 (a) and 2.11 (b) depict two possible architectures for the transformer block. In the original design, layer normalization is applied after MHA and the fully connected layer. However, prior research (Parisotto et al., 2020; Xiong et al., 2020) proposed an identity map reordering approach, positioning the layer normalization before these layers. This adjustment allows the recurrent connections to transmit an unaltered data stream.

The transformer encoder, depicted in Figure 2.11 (c), consists of a series of stacked transformer blocks. The process starts by embedding the input sequence to align with the transformer's dimensions, followed by the integration of the positional encoding.

2.4.5 Segment-Level Recurrence

The conventional transformer architecture processes sequences in fixed-length segments. When a sequence becomes too lengthy and computationally challenging, it is segmented into smaller chunks. A limitation arises when a segment is processed in isolation, devoid of context from its neighboring segments, leading to potential loss in capturing long-term dependencies.

The Transformer-XL (TrXL), "XL" denoting "extra-long" sequences, addresses this by weaving recurrent connections between successive segments. This design ensures a continuous flow of information across segment borders. Rather than recalculating hidden states for every new segment, TrXL leverages the hidden states from preceding segments. These cached states act as a contextual memory for the current segment, forging a recurrent linkage between segments. This architecture facilitates the modeling of extended

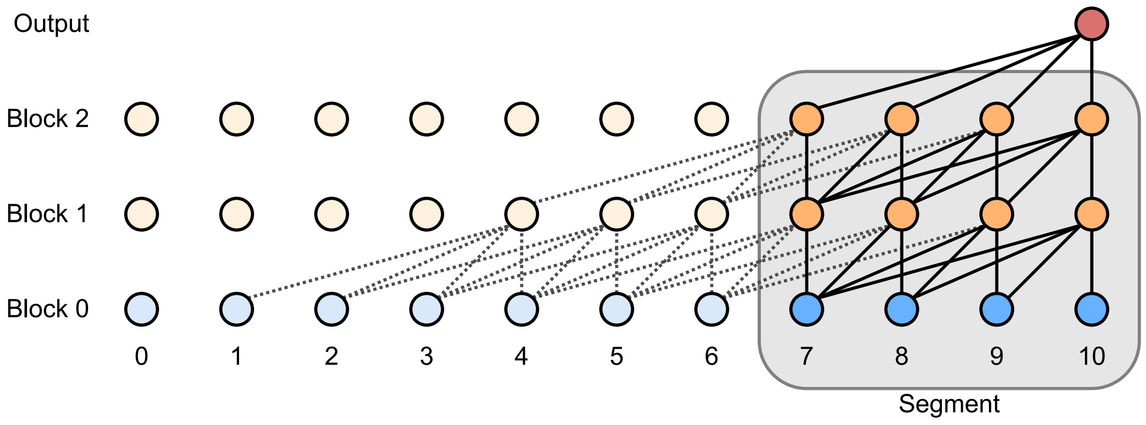


Figure 2.12: Segment-Level Recurrence in Transformer-XL. Consider a sequence-to-one task, where TrXL is composed of 3 stacked blocks and operates on a segment length of 4. Nodes represent latent representations: blue for inputs, orange for intermediary block-produced hidden states, and red for the final context vector. Bold connections back-propagate gradients, while dotted ones do not. Recurrent connections reference cached hidden states from previous segments. Here, the model can attend to information from the current time step back to time step 1. Contrarily, the original transformer only considers data within the current segment, from time step 7 to 10. By enlarging the segment length or adding another transformer block, the context length expands.

dependencies, as information can traverse through these recurrent pathways. While information flows freely, gradients do not back-propagate through earlier segments. Figure 2.12 provides a visual representation of this recurrent information flow that transcends individual segments.

2.4.6 Relative Positional Encoding

The standard transformer model employs absolute positional encodings to embed sequence order information. However, when reusing hidden states, as done in TrXL, maintaining coherent positional information becomes more challenging. Specifically, the model might find it difficult to differentiate between elements from distinct time steps that share the same absolute position. To circumvent this limitation, relative positional encodings are introduced. Rather than depending solely on the absolute position of each sequence element, this method emphasizes the relative distance between elements. This approach is more intuitive and generalizable, as it allows the model to determine the temporal order of a segment based solely on the relative distances between items (Dai et al., 2019).

For simplification, consider the unscaled dot product attention for a single attention value, with the addition of absolute positional encoding, which can be written as:

$$a_{ij} = \left((x_i + p_i)W^Q\right)\left((x_j + p_j)W^K\right)^\top \tag{2.59}$$

Here:

- x represents the input data.
- i and j are the positions of the query and the key in the sequence, respectively.
- p is a positional encoding element, retrieved from Equation 2.57.
- W^Q and W^K are the weight matrices associated with the query and key, respectively.

Dai et al. (2019) break down this formulation into four components:

$$\begin{aligned} a_{ij} = {} & (x_i W^Q)(x_j W^K)^\top + (x_i W^Q)(p_j W^K)^\top \\ & + (p_i W^Q)(x_j W^K)^\top + (p_i W^Q)(p_j W^K)^\top \end{aligned} \tag{2.60}$$

Subsequently, the appearances of the absolute positional encoding of Equation 2.60 are substituted for relative ones and learnable parameters:

$$\begin{aligned} a_{ij} = {} & (x_i W^Q)(x_j W^K)^\top + (x_i W^Q)(r_{i-j} W^K)^\top \\ & + (u^\top W^Q)(x_j W^K)^\top + (v^\top W^Q)(r_{i-j} W^K)^\top \end{aligned} \tag{2.61}$$

In this context, r_{i-j} denotes the relative positional encoding derived from the sinusoidal positional encoding (Equation 2.57), while u and v are trainable parameters. Dai et al. (2019) interpret the above formulation as follows:

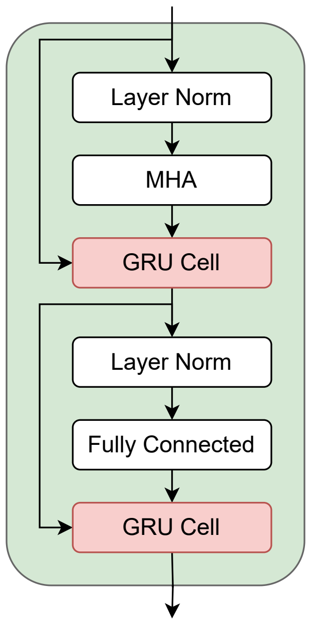


Figure 2.13: Gated Transformer-XL

- (a) signifies content-based addressing.
- (b) encapsulates a content-dependent positional bias.
- (c) represents a universal content bias.
- (d) conveys a global positional bias.

The just described relative positional encoding is linked to the family of Modifying Attention Matrix (MAM) approaches. More research is conducted on APE and MAM variants, where Dufter et al. (2022) provide a comprehensive overview.

2.4.7 Gated Transformer-XL

In DRL, Parisotto et al. (2020) pioneered the utilization of transformers as the underlying memory structure for an RL agent. They introduced the Gated Transformer-XL (GTrXL) architecture, an enhancement of TrXL, where the residual connections, associated with pre-layer normalization, are replaced by GRU cells (Section 2.3.3) as illustrated in Figure 2.13. In this refined design, the outputs from the MHA and the fully connected layer assume the role of the hidden states for the GRU cells. These former residual connections now function as the primary inputs to the GRU cells. In this setup, the recurrent cells are not unfolded. Instead, the gating mechanism is leveraged to regulate information flow.

Chapter 3

Related Work

The primary contributions of this thesis are twofold: a recurrent and a transformer-based baseline, and a new benchmark featuring finite and endless tasks. Historically, both domains have seen extensive research. This section aims to offer a thorough review of related contributions and highlight significant shortcomings:

- the absence of readily accessible and reproducible memory-based DRL baselines,
- limitations present in current benchmarks concerning strong memory dependencies and frequent memory interactions, and consequently
- the need for a standard benchmark to evaluate memory-based agents.

Initially, we provide an overview of related memory-based DRL agents and mention the benchmarks they utilize. Subsequently, the scarcity of accessible baselines is addressed.

In Section 3.2, we differentiate between benchmarks that are considered and those that are not. Afterwards, we present the desiderata, a tool designed to assess the benchmarks in our current scope, highlighting that not all environments sufficiently meet the meta criteria. Beyond these criteria, we pinpoint issues associated with strong memory dependencies and frequent memory interactions.

3.1 Memory-Based Reinforcement Learning Agents

The seminal work by Mnih et al. (2015) on mastering Atari games (Bellemare et al., 2013) using DQN marked a significant milestone in the advancement of DRL algorithms. Multiple Atari games necessitate at least a short-term memory. Consider the game Pong (Figure 3.1): a single frame displays both paddles and the ball. Given this sole frame, there is no sense of motion. To derive the velocity, a second frame is essential, while a third provides insights into acceleration. Mnih et al. (2015) employed a straightforward technique known as frame stacking to capture this temporal information. Since then, more advanced memory approaches as part of DRL algorithms emerged, which commonly employ RNNs or transformers. These are briefly described subsequently.

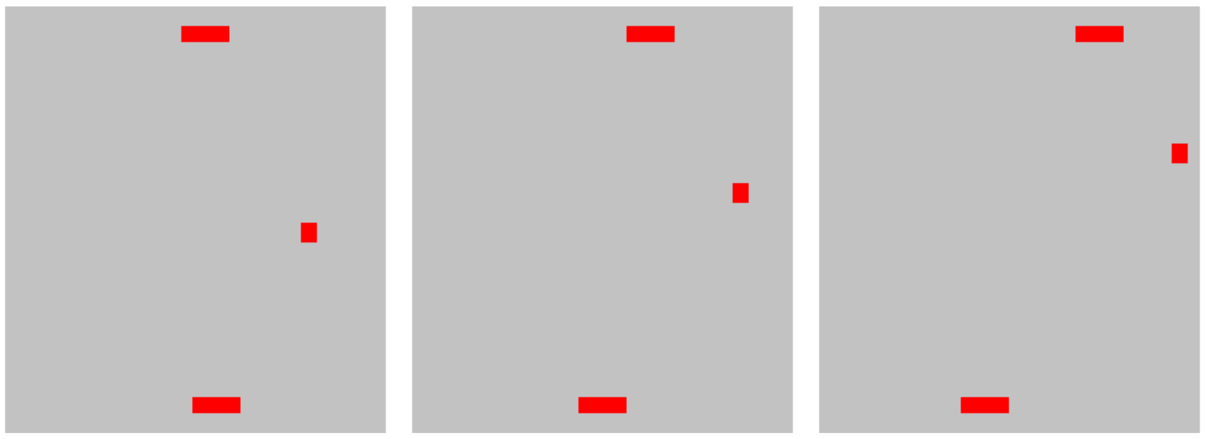


Figure 3.1: Consecutive frames in the game of Pong. A single frame does not reveal any motion information. One consecutive frame is needed to perceive velocity, while a total of three stacked frames help to identify acceleration. Feeding the agent the current and n past frames (i.e. observations) is referred to as frame stacking.

3.1.1 Overview of Current Baselines

Various memory baselines have been explored, each centered on different DRL algorithms, whether they are on-policy, off-policy, policy-based, value-based, or actor-critic. In the following sections, we chronologically and briefly describe the memory concepts, capabilities, and used benchmarks of the most prominent baselines.

Deep Recurrent Q-Network (DRQN) (Hausknecht and Stone, 2015) combines DQN with an LSTM cell. The replay buffer stores sequences, while initializing the recurrent hidden state to zero during training. The effectiveness of this approach is demonstrated on Atari games that involve frame flickering.

Asynchronus Advantage Actor-Critic (A3C) (Mnih et al., 2016) primarily emphasizes the asynchronous process of actor-learner pairs and the utilization of the advantage function. The integration of an LSTM cell as memory is not its central discussion point. However, when the agent is equipped with this memory mechanism, its performance surpasses frame stacking baselines, as DQN, in 19 of the 57 Atari games.

Importance Weighted Actor-Learner Architecture (IMPALA) (Espeholt et al., 2018) is an actor-critic algorithm designed for distributed training. In such a setting, the policy collecting training data can lag behind the policy being optimized. V-Trace addresses this by adjusting importance sampling weights to correct off-policy data. Like A3C, IMPALA employs an LSTM cell for memory, though the specifics of this integration are not elaborated upon in depth. Performance metrics on Atari games and DeepMind

Lab (DMLab-30) (Beattie et al., 2016) indicate that IMPALA outperforms A3C.

Recurrent Replay Distributed DQN (R2D2) (Kapturowski et al., 2019) extends Deep Recurrent Q-Network (DRQN) with a distributed prioritized experience replay buffer, storing overlapping sequences of fixed length. A burn-in period is utilized to address staleness in recurrent hidden states by propagating a portion of the sequence for a more effective starting state. R2D2's performance is evaluated on Atari games and DMLab-30.

Memory Recall Agent (MRA) (Fortunato et al., 2019), built on IMPALA, employs a twofold memory mechanism: a working memory based on an LSTM and a slot-based episodic memory containing summaries of past experiences. By utilizing a query-key approach, information is read from and written to the episodic memory. MRA is evaluated on the Memory Task Suite contributed by the same publication.

Gated Transformer-XL (GTrXL) (Parisotto et al., 2020) adds the concept of Transformer-XL to enable an agent, trained with V-MPO (Song et al., 2020), to leverage memory. By incorporating an identity map reordering and a gating mechanism inspired by GRU, TrXL is further improved. GTrXL's performance is demonstrated on DMLab-30, Numpad, and Memory Maze.

The **Hierarchical Chunk Attention Mechanism** (HCAM) (Lampinen et al., 2021) builds upon TrXL by partitioning the agent's memory into chunks and selecting the most relevant chunks based on their summaries to attend to their stored information in detail. HCAM is trained using IMPALA and evaluated on multiple environments including Dancing the Ballet, Object Permanence, Rapid Word Learning, Visual Match, Paired Associative Inference, and One-Shot StreetLearn Navigation.

History comprEssion via Language Models (HELM) (Paischer et al., 2022a) utilizes pre-trained language models to compress a history of observations into a single context vector (memory). It employs a frozen Transformer-XL encoder, pre-trained on the Wiki-Text 103 dataset (Merity et al., 2017). HELM is evaluated on the Procgen (Cobbe et al., 2020) and Minigrid (Chevalier-Boisvert et al., 2018) environments.

3.1.2 Lack of Accessible and Reproducible Memory Baselines

The accessibility of the memory-related approaches discussed above pose a shared concern. While IMPALA and HELM are open source, the implementations of DRQN, R2D2, A3C, MRA, and GTrXL are not public. Although HCAM is partially open source in terms of its model architecture, the method to leverage its interface for effective training remains ambiguous. The absence of open source implementations makes reproducing their results more challenging, as implementation details play a crucial role. This issue is particularly pronounced for GTrXL, as to the best of our knowledge, no one has yet been able to reproduce its underlying DRL algorithm, V-MPO.

Within the community, RLlib (Liang et al., 2018), DI-engine (DI-engine Contributors, 2021), and Brain Agent (Lee et al., 2022) advertise to feature GTrXL. However, these frameworks are large and complex, supporting multiple DRL algorithms, model architectures, and environments, which hampers verification, debugging, gaining insights, and making modifications. We tested RLlib with Memory Gym, but obtained negative returns even though the environments do not signal negative rewards. DI-Engine's GTrXL implementation is based on R2D2, and our experiments revealed poor sample throughput by a magnitude of 10 compared to our GRU baseline. When debugging Brain Agent's implementation, we noticed an unexpected discrepancy in the query length between data sampling and optimization. We expected only one query to adhere to the sequence-to-one approach, which we follow. While we do not claim that these frameworks are dysfunctional, we present our limited experience in exploring them. We believe that our contributed open source baselines are easy to follow, which facilitates reproducibility and further research in the field.

3.2 Benchmarking Memory-Based Agents

The aforementioned studies have explored the use of memory-based agents in a variety of environments. Some of them are originally fully observable but are turned into POMDP by adding noise or masking out information from the agent's observation space. For instance, this was done for the Arcade Learning Environment (Bellemare et al., 2013) by using flickering frames (Hausknecht and Stone, 2015) and common control tasks by removing the velocity information from the agent's observation (Heess et al., 2015; Meng et al., 2021; Shang et al., 2021).

Control tasks also touch on the context of Meta-Reinforcement Learning, where memory mechanisms are prominent (Wang et al., 2021; Melo, 2022; Ni et al., 2022). The same applies to Multi-Agent Reinforcement Learning (Berner et al., 2019; Baker et al., 2020; Vinyals et al., 2019). As we solely focus on benchmarking the agent's memory and its ability to generalize, we do not compare Memory Gym to environments of more complex contexts such as Obstacle Tower (Juliani et al., 2019), DeepMind (DM) Hard Eight (Gülçehre et al., 2020a), Crafter (Hafner, 2022), or DM Alchemy (Wang et al., 2021). These might need additional components to the agent's architecture and its training paradigm, for instance to approach a notable exploration challenge.

Following this section, we provide an overview of the considered related environments. Defining the desiderata towards their assessment follows. Initially, we identify shortcomings when applying the meta desiderata, which is followed by investigating whether these environments comply with strong memory dependence and frequent memory interactions. Finally, we draw the conclusion that a standard benchmark in this context is missing.

3.2.1 Overview of Current Benchmarks

Central to various memory-dependent tasks are the T-Maze (Wierstra et al., 2007) and the Morris water maze (Morris, 1981). In the T-Maze, the agent must memorize an initial goal cue, maintain this memory as it navigates an alley, and then recall the cue at the T-cross to choose the correct path and complete the task. Within the context of RL and considering the Morris water maze, the agent's objective is to find a hidden platform in a 2D space. Once found, the agent is randomly relocated and tasked with locating the platform again. Next, we provide a chronological overview of benchmarks recently employed, offering brief insights into their tasks, observation spaces, and action spaces.

DeepMind Lab (DMLab-30) (Beattie et al., 2016) presents a collection of 30 procedurally generated first-person 3D environments. Starting from a general motor-control navigation task based on a set of discrete actions, each environment poses a different goal such as collecting fruit, playing laser tag, or traversing a maze. Parisotto et al. (2020) categorize these environments into memory and reactive tasks. The agent receives visual observations and optionally depth information as well as language instructions.

VizDoom (Wydmuch et al., 2018) features first-person shooter 3D environments that revolve around motor-control navigation tasks, with the added challenge of computer-controlled enemies that may confront the agent. The agent's observation space consists of visual observations, while its action space is defined by discrete actions.

Minigrid (Chevalier-Boisvert et al., 2018) includes a 2D grid memory task inspired by the T-Maze. The action space features a discrete set of actions, where the most relevant actions are to rotate and to move forward. The agent's perception of the environment is through a frontal rectangular field of view, presented as a visual observation.

Miniworld (Chevalier-Boisvert, 2018) presents a collection of first-person 3D environments, drawing inspiration from tasks found in Minigrid. Its mechanics, observation, and action spaces parallel to some extent those observed in VizDoom and DMLab-30.

The **Memory Task Suite** (Fortunato et al., 2019) offers a variety of memory-centric environments spanning four categories: PsychLab (Leibo et al., 2018) for image-based tasks (such as change detection), Spot the Difference for cue memorization, goal navigation tasks modeled after the Morris water maze, and transitive object ordering tasks. Within these settings, the agent is prompted to execute discrete actions. It perceives its surroundings through visual observations and a game state vector that indicates its position and score.

Procgen (Cobbe et al., 2020) consists of 16 2D environments that offer procedurally generated and fully observable levels. These environments encompass a combination of puzzle-solving, platforming, and action-based games. 6 of these environments have been modified to become partially observable by reducing and centering the agent's visual observation on itself. The action space is discrete.

Numpad (Parisotto et al., 2020) requires the agent to accurately press a series of keys in a specific sequence of fixed length. The sequence itself is not provided to the agent, resulting in the agent employing a trial-and-error approach while memorizing the underlying sequence. In terms of observations, the agent is presented with a game state vector, and it can execute actions from a discrete set.

Memory Maze (Parisotto et al., 2020) shares similarities with the Morris Water maze, as the agent must locate an apple, then undergoes random repositioning, requiring it to find the apple's location again and again. This environment equips the agent with 8 continuous and one discrete action. The agent's perception is accomplished by a visual observation.

Dancing the Ballet (Ballet) (Lampinen et al., 2021) presents a sequence of up to 8 dances visually, with the agent remaining stationary. After all the dances are displayed, the agent is required to identify a specific one to successfully complete the task. While presented as grid world, the agent utilizes discrete actions, a visual observation, and a language instruction.

PopGym (Morad et al., 2023) has been published at ICLR 2023 as our original work contributing Memory Gym (Pleines et al., 2023). It is a benchmark consisting of 15 environments that are tagged as diagnostic, control, noisy, game, or navigation. These environments are designed to quickly converge by providing vector observations instead of visual ones. Most environments rely on discrete actions, while others offer a multi-discrete action space.

3.2.2 Desiderata of Memory Benchmarks

Here, we define the desiderata that we believe are essential to memory benchmarks.

Accessibility refers to the competence to easily set up and execute the environment. Benchmarks shall be publicly available, acceptably documented, and open source while running on the commonly used operating systems Linux, macOS, and Windows. Linux, in general, is important because many High-Performance Computing (HPC) facilities employ this operating system. As HPC facilities might not support virtualization, benchmarks should not be solely deployed as a docker image or similar. At last, benchmarks shall run headless because otherwise, these potentially rely on dependencies as xvfb or EGL, which HPC facilities may not support as well. Suppose relevant benchmark details, such as environment dynamics, are missing. In that case, it can be desirable to support humanly playable environments so that these can be explored in advance.

Fast simulation speeds, which achieve thousands of steps per second (i.e. Frames per Second or FPS), allow training runs to be more wall time efficient, enabling to upscale experiments and their repetitions or faster development iterations. The benchmark's speed also depends on the time it takes to reset the environment, setting up a new episode for the agent to play. Towards maxing out FPS on a single machine, benchmarks shall be able to run multiple instances of their environments concurrently.

High diversity attributes environments that offer a large distribution of procedurally generated levels to reasonably challenge an agent's ability to generalize (Cobbe et al., 2020). Notably, if diversity is absent, the environment invites the agent to overfit. Also, it is desirable to implement smoothly configurable environments to evaluate the agent at out-of-distribution levels. These levels are built using modified generation mechanisms, ensuring they deviate from those the agent trained on. For example, levels might sample gravity scales from a predefined range. Values beyond this range are considered as out-of-distribution.

Scalable difficulty seamlessly complements the concept of high diversity. It shall make environments controllable such that their current hardness can be increased or decreased. Easier environments can have benefits: a proof-of-concept state is sooner reachable during developing novel methods, and research groups can fit the difficulty to their available resources. Moreover, increasing the difficulty ensures that already competent methods may prove themselves in new challenges to demonstrate their abilities or limits. While the combination of scalability and diversity offers extensive control, the inherent complexity might daunt newcomers to the benchmark. Therefore, a clear default configuration and comprehensive documentation become imperative.

Strong memory dependence refers to tasks that are only solvable if the agent can recall past information (i.e. successfully leveraging its memory). Simply put, agents without memory are supposed to be unsuccessful when confronted with memory-dependent tasks. To ensure that the utilized memory mechanism is working, the benchmark's implementation shall minimize loopholes that hint on past environmental states or observations. Minimizing, because some details may be pivotal to the underlying task and therefore cannot be eradicated. In this favor, tasks that can be solved using short-term information, particularly through basic frame stacking, shall be omitted as well.

Strong dependency on frequent memory interactions is a criteria for tasks that forces the agent to recall information from and add information to its memory frequently. This does not apply to tasks, which are closely related to the T-Maze concept, where the agent requires only two primary interactions: storing the cue information and subsequently retrieving it to solve the task. Between these interactions, the agent's primary focus is on preserving the stored information. We believe the preservation of information over time and amidst noise is better suited, in terms of efficiency, to other paradigms, such as supervised learning.

3.2.3 Assessing Meta Desiderata

Table 3.1 provides information on the meta desiderata of the aforementioned environments. Some environments are inaccessible to some extent, as some cannot be run headless.

DM Memory Maze, DM Numpad, and DM Object Permanence are not publicly available

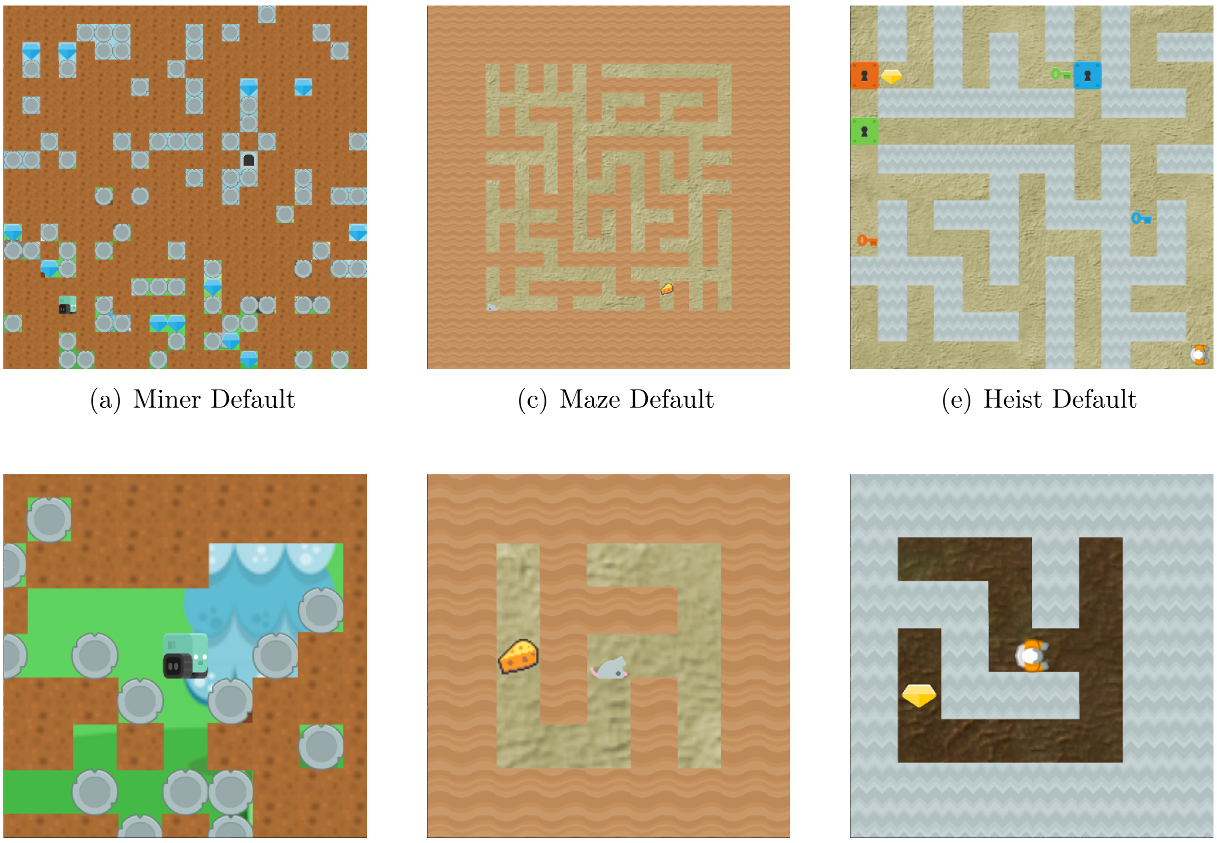
Table 3.1: An overview of the simulation speed and meta desiderata of the so far mentioned environments. The meta desiderata cover the criteria that are applicable to any benchmark in DRL. The mean FPS are measured across 100 episodes using constant actions using an AMD Ryzen 7 2700X CPU. Memory Gym and Procgen are averaged across its memory distribution environments. The measurement from PopGym is retrieved from their GitHub repository. The other benchmarks are measured using a single environment. ● refers to true and ○ to false.

Benchmark	Mean FPS	Meta Desiderata										
		Publicly Available	Open Source	Linux	macOS	Windows	Headless	Docker	Concurrent	Playable	High Diversity	Scalable Difficulty
Memory Gym (ours)	10123	●	●	●	●	●	●	○	●	●	●	●
PopGym (Morad et al., 2023)	39130	●	●	●	●	●	●	○	●	●	●	●
Procgen Memory Distribution (Cobbe et al., 2020)	18530	●	●	●	●	●	●	○	●	●	●	●
DM Ballet (Lampinen et al., 2021)	6631	●	●	●	●	●	●	○	●	●	●	●
MiniGrid Memory (Chevalier-Boisvert et al., 2018)	5185	●	●	●	●	●	●	○	●	●	○	●
MiniWorld Maze (Chevalier-Boisvert, 2018)	851	●	●	●	●	●	○	○	●	●	●	●
VizDoom (Wydmuch et al., 2018)	549	●	●	●	●	●	●	○	●	●	●	●
Crafter (Hafner, 2022)	482	●	●	●	●	●	●	○	●	●	●	○
DM Memory Task Suite (Fortunato et al., 2019)	442	●	○	●	●	●	○	●	●	●	●	●
DMLab-30 (Beattie et al., 2016)	433	●	●	●	○	○	○	○	●	●	●	●
DM Alchemy (Wang et al., 2021)	308	●	○	●	●	●	○	●	●	●	●	○
DM Hard Eight (Gülçehre et al., 2020b)	160	●	○	●	●	●	○	●	●	●	●	○
Obstacle Tower (Juliani et al., 2019)	43	●	●	●	●	●	○	○	●	●	●	●
DM Numpad (Humplik et al., 2019)		○										
DM Memory Maze (Parisotto et al., 2020)		○										
DM Object Permanence (Lampinen et al., 2021)		○										

and can, therefore, not be used to reproduce and compare results in adjacent research. DM Memory Task Suite, DM Alchemy, and DM Hard Eight are not open source and can only be accessed via the virtualization software docker. It may be for these reasons that those benchmarks are not widely used yet.

However, DMLab-30 has more uses, but the available environments, especially those used in the context of memory, are superficially documented and thus lack vital information on their dynamics and more. Parisotto et al. (2020) divide these environments into reactive and memory tasks without providing further reasoning or evidence. Potential users have to explore these environments by playing them, which is possible on machines running Linux only.

Procgen, DM Ballet, MiniGrid, PopGym, and Memory Gym (ours) are the only environments that achieve a throughput of thousands of steps per second. By far, PopGym is the fastest environment, which is plausible, because it relies only on game state vector


(a) Miner Default (c) Maze Default (e) Heist Default

(b) Miner Memory (d) Maze Memory (f) Heist Memory

Figure 3.2: Three Procgen environments and their memory views: top row shows default settings, and the bottom depicts the agent's memory mode perspective. From left to right: Miner has a cart-agent collecting 12 diamonds; Maze features a mouse-agent seeking cheese; Heist expands on Maze, adding key-door puzzles to reach a golden diamond.

observations, omitting expensive frame rendering. Rendered 3D environments usually achieve a smaller throughput, as VizDoom or the DM Memory Task Suite.

3.2.4 Assessing Strong Memory Dependence

It is challenging to discern the efficacy of a memory mechanism if an environment can be somewhat solved by an agent without memory. This dilemma is evident in four of the six environments from Procgen's memory distribution: Miner, Maze, Heist, CaveFlyer, Dodgeball, and Jumper.

In the maze-like terrains of Jumper and CaveFlyer, agents could potentially benefit from memory. These environments present vast expanses that agents must traverse to reach a goal, all while dodging hazardous obstacles. Given the complexity and the potential need for hundreds of steps to achieve success, memory can be advantageous. An agent with effective memory utilization might navigate more efficiently, avoiding repeated visits to the same dead-ends.

Conversely, the environments of Miner, Maze, and Heist pose concerns. In Miner, as illustrated in Figure 3.2 (a), the agent's objective is to gather 12 diamonds before finding an exit. A significant challenge arises from the boulders. If an agent digs out the dirt beneath a boulder, it falls, potentially striking the agent and ending the episode. The absence of dirt and the boulder's position serve as historical cues. For instance, a region with less surrounding dirt, as shown in Figure 3.2 (b), indicates the agent's prior presence. The agent's tendency to move upwards when encountering multiple boulders at the environment's bottom ground further underscores this.

Both Maze and Heist have levels that are entirely observable, as highlighted in Figures 3.2 (d) and 3.2 (f). Given these characteristics, it is unsurprising that Paischer et al. (2022a) found comparable performances between memory-based and memory-free approaches. A similar pattern is observed in Dodgeball, though we will not delve deeper into this.

PopGym features environments that provide similar issues. The game environment Concentration works like the toddler game "memory". By flipping face down cards, the agent needs to find pairs. Overtime, the problem gets easier as card pairs are removed form the game upon unveiling a match. We believe this to be solvable by a memory-less agent, while being less efficient. This is supported by the similar scores that Morad et al. (2023) report. A similar impression, based on the reported results, can be retrieved from the Battleship environment.

Because of these issues, how can one reliably determine whether the implemented memory mechanism is working or not?

3.2.5 Assessing Frequent Memory Interactions

Concerning frequent memory interactions, we believe that MiniGrid Memory, Spot the Difference (DM Memory Task Suite), and DM Ballet are not well suited to the need for memory in sequential decision-making. These environments demand the agent's memory to solely memorize the to-be-observed goal cues during the very beginning of the episode. Once the cues are memorized, there is no need to manipulate the agent's memory further. Simply maintaining it is enough to eventually recall the content once to solve the underlying task. It can be hypothesized that the extracted features from observing the goal cues are sufficient to solve the ballet environment.

To show this, the ballet environment can be made fully observable by feeding the entire sequence of cues to an RNN, functioning as an encoder that extracts a context vector that is later utilized by the agent's policy. Consequently, the policy does not need to posses memory as illustrated in Figure 3.3. The task at hand gets much easier because the agent does not need to make obsolete decisions while observing the sequence of cues as its position is frozen.

Such tasks would only challenge the capacity and robustness to noise of the agent's memory. We believe this can be done more efficiently by other benchmarks that do not belong to the context of DRL, like Long Range Arena (Tay et al., 2021).

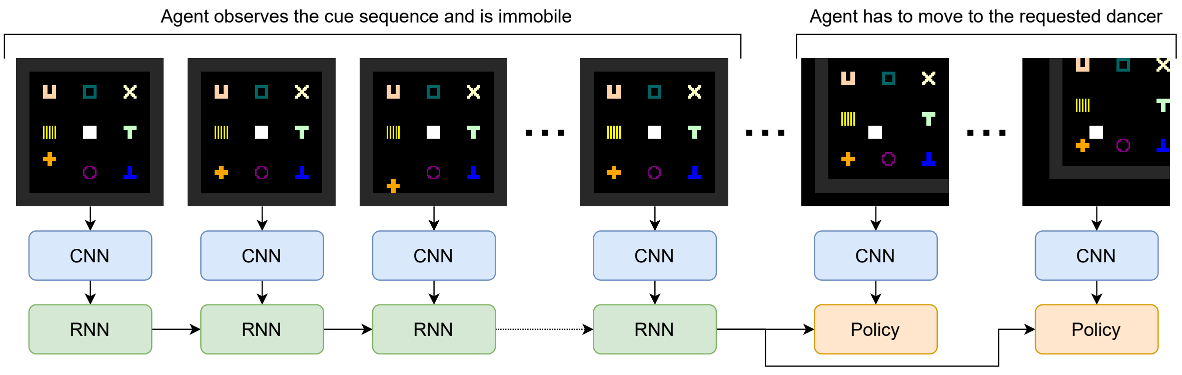

Agent observes the cue sequence and is immobile
Agent has to move to the requested dancer
CNN
RNN
Policy

Figure 3.3: DM Ballet made fully observable. The white square represents the agent amidst 8 dancers. The task involves observing sequential dances, as shown by the 4 frames with the moving orange cross, and then approaching the requested dancer. While dancing, the agent normally makes inconsequential decisions. This can be omitted by processing these frames by an encoder comprising a CNN and RNN. Then, the derived and fixed context vector (final RNN output) guides the agent's actions to solve the task, eliminating the need for memory in its policy.

3.3 Key Insights: Absent Standard Benchmark

When skimming the approaches described in Section 3.1.1, it appears that there is a lack of consensus regarding the choice of environments for evaluation as they assess their approaches on different benchmarks, thereby hindering comparisons between them. It maybe for the reasons that we previously explored:

- Procgen consists of environments that do not strongly depend on memory.
- DM Ballet, Spot the Difference (Memory Task Suite), and MiniGrid Memory do not require frequent memory interactions.
- DM Numpad, Memory Maze, and Object Permanence are unavailable.
- The Memory Task Suite is only available through virtualization.
- DMLab-30 is purely documented, does not run headless, and only runs on Linux.
- Other 3D environments suffer from low simulation speeds.

PopGym (Morad et al., 2023) shares some of these issues, while several environments lack novelty. For instance, its maze tasks have existed before (Cobbe et al., 2020; Parisotto et al., 2020). The same applies to the control tasks where information is masked out from the game state vector or noise perturbations are added (Heess et al., 2015; Meng et al., 2021; Shang et al., 2021). The most promising environment of PopGym seems to be Minesweeper, where memory baselines outperform a memory-less agent (Morad et al., 2023).

All those aforementioned issues make it difficult to select a benchmark that adheres to our defined criteria. Clearly, a comprehensive standard benchmark is lacking. Although it still needs to be discussed what a standard benchmark should comprise. This discussion is pursued in Section 7.4.

3.4 Key Insights: Inaccessible Memory Baselines

The essence of Section 3.1 is that memory in DRL agents mostly relies on RNNs or transformers. Unfortunately, many of the research advancements are not shared openly through open-source code, making it tough for others to replicate or build upon them. This is especially pronounced for GTrXL, where no one has yet been able to implement its main training method called V-MPO. Among all the discoveries, only the authors behind IMPALA and HELM have fully shared their source code. The community has made attempts to equip agents with transformer-based memory capabilities. However, we struggled to leverage these largely grown frameworks effectively.

On a brighter note, numerous community implementations are available for agents utilizing RNNs. The most notable one is the single file implementation by CleanRL (Huang et al., 2022b,a), which combines PPO with an LSTM. The details behind it were published concurrently with our GRU baseline implementation (Pleines et al., 2022).

Chapter 4

Memory Gym Environments

This chapter introduces Memory Gym, a novel benchmark dedicated to address the shortcomings of existing benchmarks as detailed in Section 3.3. Memory Gym is not only open-source[1] and easily accessible[2], but it also features fast simulation speeds (Table 3.1), diversity, and scalability. By harnessing the power of procedural content generation (Shaker et al., 2016), we achieve both scalability and diversity, while there is a large interface to easily modify the environments' difficulties. All environments strongly depend on memory and frequent memory interactions. The empirical evidence supporting the former claim can be found in Section 6.2.1, while the latter is proven in this chapter.

At its core, Memory Gym encompasses three unique environment families: Mortar Mayhem, Mystery Path, and Searing Spotlights. Each family presents both a finite task[3], which concludes once its objective is met, and an endless task, which continues as long as the agent performs successfully. However, if the agent acts poorly, potentially infinite episodes terminate. To our knowledge, endless tasks, especially concerned with memory challenges, do not exist yet.

It is important to clarify that "endless" should not be confused with "open-ended". Memory Gym's endless environments are not open-ended in nature; they possess clear objectives and defined structures. In contrast, open-ended environments offer agents freedom to develop unique skills and strategies without stringent constraints as depicted by platforms as Minecraft (Fan et al., 2022).

This chapter is structured as follows: We start by drawing a parallel between endless memory-dependent tasks and the classic car game "I packed my bag", which inspired our approach. We then detail shared environment properties, including observation and action spaces. Each environment family is explored in sequence, emphasizing both finite and endless versions. We introduce reset parameters for modifying and scaling stochastic environment levels and conclude by comparing our environments with related ones in the field. Then, we offer insights into the episode lengths for the finite versions. Finally, we delve into the finite tasks again to emphasize their demand for frequent memory write and read interactions.

[1]Source Code: `https://github.com/MarcoMeter/endless-memory-gym`

[2]PyPI Package: `https://pypi.org/project/memory-gym/` Accessed: 2023-09-18

[3]Whenever we refer to Mortar Mayhem, Mystery Path, or Searing Spotlights, it is in the context of their respective finite tasks.

4.1 The Concept of Endless Memory Tasks

The endless concept can be grasped by following a narrative about "I packed my bag":

> Imagine embarking on a long-awaited family vacation, with the open road stretching ahead. As the car hums along, a lively atmosphere envelops the passengers, young and old alike, as they engage in a playful game of "I packed my bag". Each family member takes turns adding an item to an imaginary bag, attempting to remember and recite the growing list correctly. However, as the game unfolds and the list lengthens, the passengers encounter mounting challenges illustrating the finite nature of their individual memories. Despite best efforts, varying strategies, and different capabilities, the weight of the growing list eventually leads to resignation.

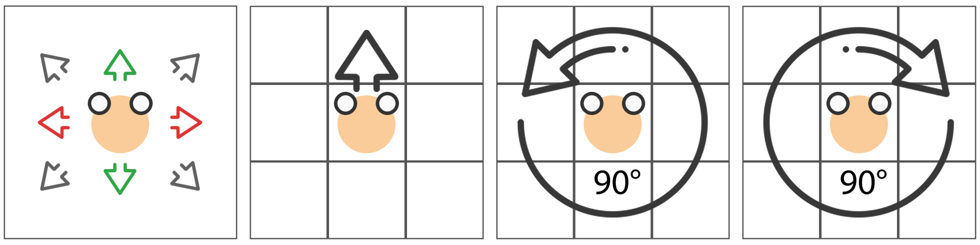
Due to the game's ever-growing or endless concept, it serves as a revealing test of memory effectiveness, showcasing the limitations that decision-makers face when confronted with the increasing demands of retention and recall. Our focus is on effectiveness because, in the context of finite tasks, the performance of successful agents is typically compared by their efficiency. Efficiency might refer to agents that require fewer agent-environment interactions or less wall-time. However, the endless framework primarily highlights varying degrees of effectiveness over efficiency. Such a perspective offers insights into the distinct capabilities of memory-based agents. By effectiveness, we refer to achieving higher scores, which is linked to stronger memory capabilities.

4.2 Shared Environment Properties

The environments within Memory Gym are implemented using Python, PyGame, and Gymnasium (Towers et al., 2023). Their design draws inspiration from the mini-games found in the commercial video game, Pummel Party[4]. All these environments possess a common observation space. Agents perceive their surroundings through a top-down visual observation, capturing an area of 84×84 RGB pixels. In the default scenarios, additional modalities as language are not incorporated, ensuring compatibility with widely used agent and model architectures.

The action space available to the agent is multi-discrete, spanning two dimensions, each of size three, as illustrated in Figure 4.1 (a). Simpler grid-based locomotion variants, based on a single discrete action space, are also incorporated in variants of Mortar Mayhem and Mystery Path, as depicted in Figures 4.1 (b) to 4.1 (d).

[4]Pummel Party: `http://rebuiltgames.com/` Accessed: 2023-09-18



(a) Multi-Discrete (b) Move Forward (c) Rotate Left (d) Rotate Right

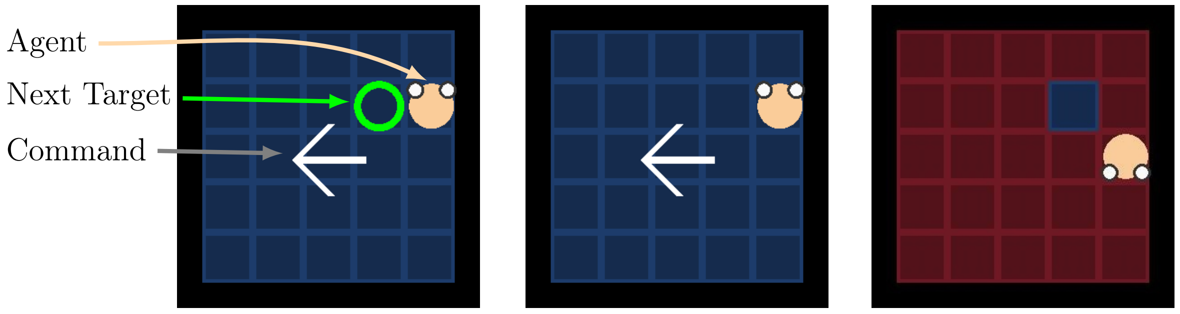
Figure 4.1: The skin-colored agent in Memory Gym and its action spaces. The agent utilizes a multi-discrete action space (a) with distinct dimensions for horizontal (red arrows) and vertical (green arrows) velocity. Both dimensions include a no-op action, enabling movement in eight distinct cardinal directions (all arrows). The agent maintains a fixed speed of 3 pixels per step, with its forward direction determined by the combined horizontal and vertical velocity. The agent's forward direction is visually represented by two white-colored circles, depicting hands. Additionally, Mortar Mayhem and Mystery Path offer a grid-like locomotion system with a discrete action space of four actions: moving forward one tile at a time (b), rotating by 90 degrees left (c) or right (d), and a no-op action.

4.3 Mortar Mayhem

Mortar Mayhem is an environment where agents memorize and execute commands in sequence. This section unpacks its design, from finite to endless variants, its scaling mechanisms, and its similarities to related environments.

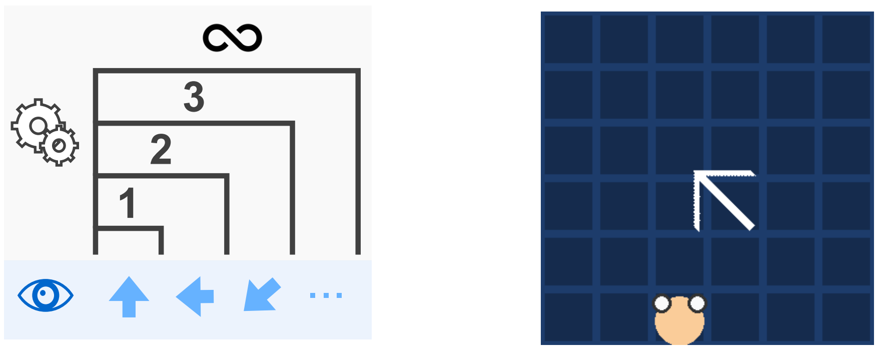
4.3.1 Finite Mortar Mayhem

Mortar Mayhem (Figure 4.2) takes place inside a grid-like arena and consists of two memory-dependent tasks. Initially, the agent is immobile and must memorize a sequence of ten commands (Clue Task), followed by executing each command in the observed order (Act Task). A command instructs the agent to move to an adjacent floor tile or remain at the current one. Failure to execute a command results in episode termination, whereas each successful execution yields a reward of +0.1. Mortar Mayhem can be simplified to solely provide the act task, where the command sequence is fully observable as a one-hot encoded game state vector. The act task necessitates frequent memory utilization since it must discern which commands have been executed to fulfill the subsequent ones, while considering a maximum of nine target tiles. To overcome this challenge, the agent can learn to track time (e.g. count steps) where memory is needed.



(a) Annotated Ground Truth (b) Agent Observation (c) Visual Feedback

Figure 4.2: Mortar Mayhem's relevant entities are depicted in the annotated ground truth of the environment (a). The ground truth includes a green circle that designates the next target tile to move to. At the beginning of an episode, the commands are sequentially rendered onto the agent's observation (b) while the agent remains immobile. Once all commands are displayed, the agent must navigate to the target tile within a specified time frame. Upon completion of the current command's execution time, the agent's success is evaluated, as illustrated in figure (c). This visual feedback is perceptible to the agent. Standing on red-colored tiles means failure. Following a brief delay of a few steps during evaluation, the agent can proceed to the next command unless the episode has terminated due to failure or completion of the entire command sequence.



(a) Observing and Executing Commands (b) Agent Observation

Figure 4.3: In Endless Mortar Mayhem, the visualization and execution of commands are alternated (a) following the idea of an automatic curriculum. Commands (blue) are visualized only once in-between executing a new command and the previous ones. The arena (b) seamlessly occupies the entire screen, enabling the agent to reenter from the opposite side if it walks off.

4.3.2 Endless Mortar Mayhem

Endless Mortar Mayhem (Figure 4.3) introduces a unique concept inspired by the game "I packed my bag". It extends the to-be-executed command sequence continuously, enabling potentially infinite sequences. Two significant modifications are made to Mortar Mayhem to accommodate this feature. In the previous setup, the agent had to observe the entire sequence before executing it. However, in the endless task, we adopt a different approach by alternating between command visualization and execution.

As shown in Figure 4.3 (a), the agent only perceives one command during visualization. During execution, the agent must execute all previous commands along with the new one. An episode begins by displaying one command, followed by its execution. Subsequently, the next visualization reveals only the second command, requiring the agent to execute both the first and second commands. This automatic curriculum allows the to-be-executed command sequence to continually expand, while each new command is presented only once. A reward of +0.1 is signaled for every single successful command.

The second modification concerns the arena's boundaries. In Mortar Mayhem, the agent's position would need to be reset after command execution due to certain commands being unavailable when the agent is adjacent to the arena's boundaries. In Endless Mortar Mayhem, we eliminate the borders and seamlessly scale the arena to the screen's dimensions. Additionally, we implement a screen wrap feature that transforms the arena into a torus. Consequently, if the agent walks off the screen, it reappears on the opposite side without requiring relocation. Finally, the agent then executes the sequence of commands again from its new position.

4.3.3 Scaling Mortar Mayhem

Mortar Mayhem's difficulty can be scaled by several parameters (Table 4.1), such as the number of commands, the show duration of a single command, and the delay between showing commands. Furthermore, the time allocated for executing a single command can be modified. These options can also be sampled upon resetting an episode. To simplify the difficulty, the agent can be equipped with grid-like movement, allowing it to move one tile at a time. Doing so, the agent is not commanded to move to diagonal tiles.

Concerning Endless Mortar Mayhem, it is an auto-curriculum mechanism in itself. The hardness scales with the agent's progression.

4.3.4 Comparing Mortar Mayhem to Related Work

While Endless Mortar Mayhem is unmatched by adjacent work, the finite variant bears similarities to Dancing the Ballet (Lampinen et al., 2021) and PopGym's Autoencode (Morad et al., 2023). In each of these environments, there's an initial cue memorization phase during which the agent remains stationary. However, while DM Ballet necessitates only a single decision during the action phase, both Autoencode and Mortar Mayhem demand a sequence of accurate decisions. A unique aspect of Autoencode is its requirement

Table 4.1: Reset Parameters for Mortar Mayhem. Parameters marked with an asterisk (*) are sampled from the provided set "[...]". The to-be-sampled sets usually default to only one element. While Mortar Mayhem Grid retains most options, it modifies three of them. Endless Mortar Mayhem removes two parameters of its predecessor: the episode success reward and the predefined number of commands.

Mortar Mayhem	
Parameter	**Default**
Agent Scale	0.25
Agent Speed	3
Arena Size	5
No. Available Commands	9
No. Commands*	[10]
Command Show Duration*	[3]
Command Show Delay*	[1]
Execution Delay*	[6]
Execution Duration*	[18]
Show Visual Feedback	True
Reward Command Failure	0
Reward Command Success	0.1
Reward Episode Success	0

Mortar Mayhem Grid	
Parameter	**Default**
Execution Delay*	[2]
Execution Duration*	[6]
No. Available Commands	5

for the agent to respond to cues in reverse order, meaning the first cue observed becomes the last one to be acted upon. Conversely, in Mortar Mayhem, the intervals between cues and their corresponding actions remain consistent. This particular mechanism shall be included optionally in upcoming releases. Additionally, Mortar Mayhem offers the flexibility to sample different delays and durations, a feature adopted from theDM Ballet environment.

4.4 Mystery Path

Mystery Path offers a unique blend of navigation and memory challenges. Agents traverse hidden paths, adapting based on past successes and failures. This section provides a comprehensive overview of the environment, detailing its mechanics, variations, and how it stands in comparison to other memory benchmarks.

4.4.1 Finite Mystery Path

Mystery Path (Figure 4.4) challenges the agent to traverse an invisible path within a grid-like level of dimension 7×7. If the agent deviates from the path and thus falls down the pit, the agent is relocated to its origin during the next step. The episode terminates when the agent reaches the goal or exhausts its time budget of 512 steps. Reaching the goal yields a reward of $+1$. If a dense reward function is desired, the agent can be rewarded by 0.1 for visiting previously unexplored tiles. To effectively navigate the environment, the agent must frequently memorize and remember its steps on the path as well as the locations where it fell off.

The invisible path is procedurally generated using the A^* path-finding algorithm (Hart et al., 1968). Initially, the path's origin is randomly sampled from the grid's border, while the goal is placed on the opposing side in a random position. Instead of aiming for the shortest path, the cost function of A^* is sampled to generate a diverse range of paths. The path generation process randomizes some cells as untraceable (A^* walls), which adds variation to the generated paths. It is important to note that these walls are considered pits rather than physical obstacles.

4.4.2 Endless Mystery Path

Endless Mystery Path (Figure 4.5) introduces a never-ending path for the agent to traverse. Based on its original environment, we make several modifications to the path generation, the agent's observation, and the terminal conditions.

The path generation in this endless task (Figure 4.5 (a)) involves concatenating path segments. Each segment is generated as done in Mystery Path, with the only difference being that the origin of a segment is based on the final tile of the previous segment. To ensure that path tiles do not have multiple neighbors, an additional tile is added between segments. The path is always generated from left to right. Consequently, we eliminate the agent's ability to move left.

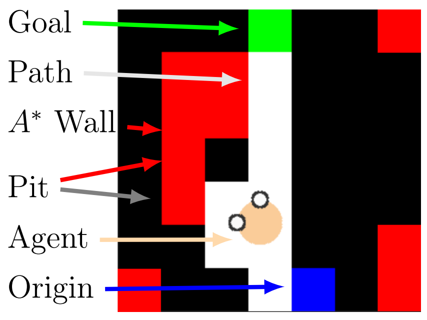
As the endless path cannot be fully captured in an 84×84 RGB pixel observation, we visually fix the agent's horizontal position. If nothing but the agent is shown in the observation, the agent lacks information about its horizontal motion. To address this, we render the path tiles behind the agent (Figure 4.5 (a)), providing visual cues of the local horizontal position.

Regarding terminal conditions, we aim to shorten episodes if the agent performs poorly. To achieve this, we implement three terminal conditions. Firstly, the agent is given a time budget of 20 steps to reach the next tile. Moving in the wrong direction or taking too long to progress within the budget leads to episode termination. The time budget is reset when the agent moves to the next tile or falls off and is relocated to the path's origin. The second condition terminates the episode if the agent falls off before reaching its best progress. Lastly, the episode is terminated if the agent falls off at the same location for a second time. With these conditions in place, an episode can potentially last indefinitely if the agent consistently succeeds.

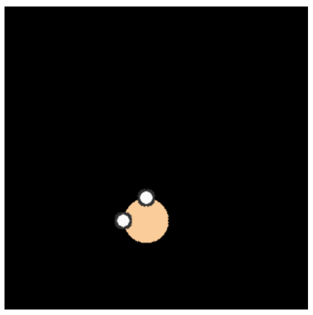
In terms of rewards, the agent earns 0.1 for each previously unvisited tile of the path it reaches. While this may initially appear as a dense reward function, any misstep that causes the agent to fall off means it cannot reclaim those rewards. The agent must retrace its steps and regain its prior progress to continue accumulating rewards. The longer it takes to catch up, the sparser the rewards become.

4.4.3 Scaling Mystery Path

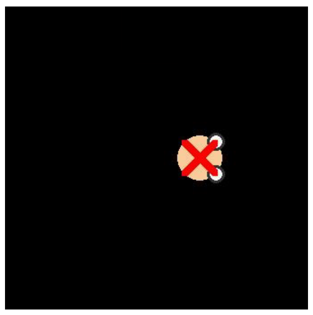
Mystery Path can be simplified to a grid variant called Mystery Path Grid, which features a single discrete action space of four actions. In this variant, the agent's time limit is



(a) Annotated Ground Truth

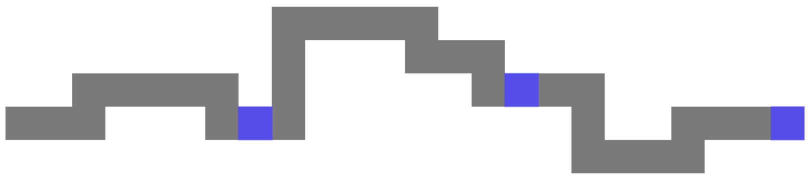


(b) Agent Observation

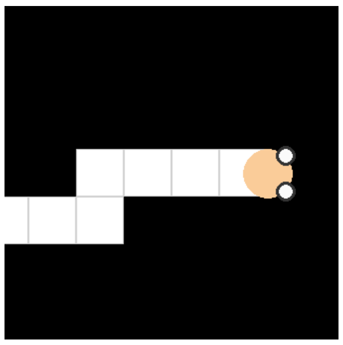


(c) Visual Feedback

Figure 4.4: The annotated ground truth depicts the relevant entities in Mystery Path (a). The walkable path is distinguished by its blue (origin), green (goal), and white color (path). Areas outside the path are designated as pits, represented by black and red colors, causing the agent to fall off and be relocated to the path's origin on the next step. The initial randomly chosen red tiles (A^* Walls) are only considered during path generation. The agent's observation includes just itself (b). If the agent deviates from the path, a red cross provides visual feedback (c).



(a) Endless Path



(b) Agent Observation

Figure 4.5: In Endless Mystery Path, the path (a) is established by concatenating path segments, following a similar generation process as in Mystery Path. To avoid multiple neighboring path tiles, we add a transition tile (blue) between segments. Due to the constraints of representing an endless path within the observation space (b), we opt to visually fix the agent's horizontal position. By rendering the path tiles behind the agent, the agent gains a sense of its horizontal motion, allowing it to perceive its progress along the path. Its horizontal position is adjustable, potentially obscuring more of the past path to increase the challenge.

Table 4.2: Reset Parameters for Mystery Path. Parameters marked with an asterisk (*) are sampled from the provided set "[...]". Mystery Path Grid overrides two parameters of its non-grid variant. Endless Mystery Path adds further configurations.

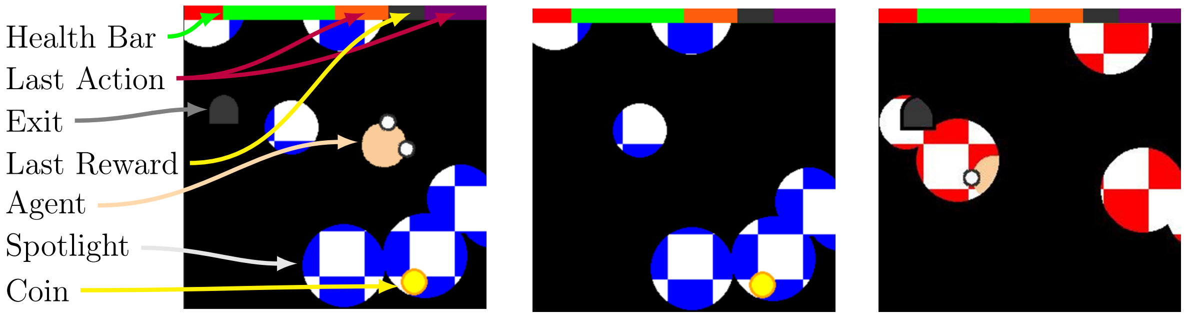
Mystery Path		**Mystery Path Grid**	
Parameter	**Default**	**Parameter**	**Default**
Max Episode Length	512	Max Episode Length	128
Agent Scale	0.25	Reward Path Progress	0
Agent Speed	3	**Endless Mystery Path**	
Cardinal Origin Choice*	[0, 1, 2, 3]	**Parameter**	**Default**
Show Origin	False	Stamina Level	20
Show Goal	False	Show Stamina	False
Show Visual Feedback	True	Show Background	False
Reward Goal	1	Show Past Path	True
Reward Fall Off	0	Camera Offset Scale	5
Reward Path Progress	0.1		
Reward Step	0		

reduced to 128 steps. Furthermore, Mystery Path can be made easier by including the path's goal and origin in the agent's observation. On the other hand, increasing the difficulty can be achieved by reducing the agent's scale, decreasing its speed, or enlarging the path. Table 4.2 depicts all available options.

Due to the ever-growing path, Endless Mystery Path poses an automatic-curriculum as Endless Mortar Mayhem. To further add difficulty in this environment, the visual horizontal position can be shifted further to the left, obscuring more information about the actual path. Rather than aiding the agent by displaying the previous path, this feature can be disabled. Instead, a grid-like background can be rendered to provide context for horizontal movement in the absence of the past path information.

4.4.4 Comparing Mystery Path to Related Work

Mystery Path closely resembles navigation principles found in environments inspired by the Morris water maze (Morris, 1981). If the agent falls off, it must return to the point just before the fall and then take alternative actions to advance on the hidden path. This process, often repeated multiple times, distinguishes Mystery Path from other environments, such as the goal navigation environment from the Memory Task Suite (Fortunato et al., 2019). A related task, as posed by Endless Mystery Path, is not apparent in adjacent research.



(a) Annotated Ground Truth (b) Agent Observation (c) Visual Feedback

Figure 4.6: The environment's annotated ground truth (a) reveals information about all relevant entities in Searing Spotlights. The top rows of pixels display the agent's remaining health points, its last action, and indicate whether a positive reward was received during the last step. The last action is encoded using two chunks and three colors, while the last positive reward is represented by two colors. Yellow circles indicate coins, while the door-shaped icon represents the exit. When all coins are collected, the exit turns green. The floor of the environment is checkered blue and white. In the agent's observation (b), the spotlights play a role in revealing or hiding other entities. As an additional form of visual feedback (c), the blue floor tiles turn red if a spotlight detects the presence of the agent.

4.5 Searing Spotlights

Searing Spotlights immerses the agent in a dark environment, where it must dodge hazardous spotlights, collect coins, and locate exits. This section explores the dynamics in this environment, its endless variant, its scalability in difficulty, and related tasks.

4.5.1 Finite Searing Spotlights

Searing Spotlights (Figure 4.6) presents a pitch-black environment to the agent, where information is revealed only through roaming and threatening spotlights. The agent starts with a limited number of health points (e.g. 5) and loses one health point per step if hit by a spotlight. The episode terminates if the agent runs out of health points or exceeds the time limit of 256 steps. To evade the imminent threats, the agent must strategically hide in the darkness. In order to escape from the encroaching spotlights, the agent needs to remember its past actions and at least one previous position to deduce its current location. This requires the agent to carefully manage its health point budget to ascertain its location, ideally with minimal exposure. As part of its observation space, the agent can perceive its last action, its current health, and whether it received a positive reward during the previous step.

To avoid monotonous behavior, two additional tasks are introduced in the environment, requiring the agent to actively traverse the environment. The first task asks the agent to collect a predetermined number of coins, while the second task involves reaching an exit to finish the episode once all the coins have been collected. Both the exit and the coins are randomly positioned during the environment's reset phase, while our implementation ensures that no entity overlaps. Collecting a coin rewards the agent with +0.25, while successfully utilizing the exit grants a reward of +1. By utilizing its memory effectively, the agent shall be able to deduce its current location and recall the positions of the coins and the exit.

4.5.2 Endless Searing Spotlights

Based on just a few modifications, we extrapolate the concept of Searing Spotlights to Endless Searing Spotlights. The exit task is removed, and the coin collection mechanism is revamped to spawn a new coin each time one is collected. This coin remains visible for six frames, and the agent has 160 steps to collect it. Additionally, the agent's health points are increased from five to ten. Unlike the original task, spotlights now appear at a consistent rate, rather than increasing in frequency over time.

While Endless Searing Spotlights unveils performance gaps between the contributed GRU and TrXL baselines (Section 6.3), the task still offers untapped potential, which we discuss in Section 7.3.3.

4.5.3 Scaling Searing Spotlights

A simplified grid version of Searing Spotlights is not available. Table 4.3 lists all customizable options. By default, episodes begin with full visibility, which gradually diminishes as the global light dims. To heighten difficulty, the agent's health points can be decreased, or the number, size, and speed of the spotlights can be scaled. The number of coins is increasable to extended the coin collection task.

4.5.4 Comparing Searing Spotlights to Related Work

Searing Spotlights uniquely challenges an agent to remain hidden in darkness, relying on memory of past positions and actions to deduce its current location. While this specific concept appears novel, the additional tasks of coin collection and finding an exit are reminiscent of tasks in other environments, as, for example, Miner (Cobbe et al., 2020) and some within DMLab-30 (Beattie et al., 2016).

4.6 Quantification of Finite Episode Lengths

This section provides insights into the quantities of episode lengths in the finite tasks of Memory Gym. When dealing with DL tasks in general, it is quite useful to know the structure of the data used for optimization. For time-series data, knowing the sequence

Table 4.3: Reset Parameters for Searing Spotlights. Parameters marked with an asterisk (*) are sampled from the provided set "[...]" or range "(...)". Endless Searing Spotlights removes all parameters concerned with spawning spotlights, while adding a single parameter to set the constant spawn rate.

Searing Spotlights			
Parameter	**Default**	Spotlight Spawn Threshold	10
Max Episode Length	256	Spotlight Radius*	(7.5-13.75)
Agent Scale	0.25	Spotlight Speed*	(0.0025-0.0075)
Agent Speed	3	Spotlight Damage	1
Agent Always Visible	False	Light Dim Off Duration	6
Agent Health	5	Light Threshold	255
Sample Agent Position	True	Show Visual Feedback	True
Use Exit	True	Show Last Action	True
Exit Scale	0.5	Show Last Positive Reward	True
Exit Visible	False	Render Background Black	False
Number of Coins*	[1]	Hide Checkered Background	False
Coin Scale	0.375	Reward Inside Spotlight	0
Coin Always Visible	False	Reward Outside Spotlights	0
No. Initial Spotlight Spawns	4	Reward Death	0
No. Spotlight Spawns	30	Reward Exit	1
Spotlight Spawn Interval	30	Reward Coin	0.25
Spotlight Spawn Decay	0.95	Reward Max Steps	0

lengths can offer a glimpse into the scale of the task. Typically, the training data and thus the sequence lengths remain stationary. However, in RL tasks, where training data is dynamically generated by an evolving policy, episode durations can fluctuate significantly.

In Mortar Mayhem, the length of an episode is influenced by the agent's skill level. As the agent becomes more adept, episodes tend to extend, but only up to an upper bound (i.e., the maximum episode length). The equations below outline how the maximum and minimum episode lengths for Mortar Mayhem, with respect to their default configuration, are calculated:

$$\begin{aligned}
\text{Clue Task} &= (\text{Show Duration} + \text{Show Delay}) \times \text{Command Count} \\
\text{Execution Length} &= (\text{Execution Duration} + \text{Execution Delay}) \times \text{Command Count} \\
\text{Act Task} &= \text{Execution Length} - \text{Execution Delay} \\
\text{Max Episode Length} &= \text{Clue Task} + \text{Act Task} \\
\text{Min Episode Length} &= \text{Clue Task} + \text{Execution Duration}
\end{aligned} \tag{4.1}$$

Table 4.4 details the lower and upper bounds for several variants of Mortar Mayhem.

Unlike Mortar Mayhem, episodes in Mystery Path and Searing Spotlights tend to

Table 4.4: Episode boundaries, measured in steps, in various instances of Mortar Mayhem.

	Mortar Mayhem	Mortar Mayhem Act	Mortar Mayhem Grid	Mortar Mayhem Act Grid
Lower Bound	58	18	46	6
Upper Bound	274	234	118	78

Table 4.5: Episode length measurements in Mystery Path and Searing Spotlights. Concerning Mystery Path, results are based on training a memory-less agent with full path visibility. Due to the agent's stochastic policy, the results may be slightly inaccurate. From 1000 episodes, each repeated 5 times, the minimum value was selected from the repetitions for each episode. Out of these, one episode failed and an outlier with 81 steps was excluded. For Searing Spotlights, the optimal episode length is determined by summing the distances from the agent to the coin and from the coin to the exit, then dividing by the agent's speed and rounding up.

	Mystery Path	Mystery Path Grid	Searing Spotlights
Min	22	7	14
Max	43	22	63
Mean	29.03	13.37	33.36
Std	4.88	3.16	7.56
Upper Bound	512	128	256
Samples	1000	1000	100,000
Policy	Agent	Agent	Optimal

be shorter as the agent refines its skills. These tasks are more flexible in structure. For instance, Mystery Path can have varying path lengths, while the distances between entities in Searing Spotlights can differ. Hence, episode lengths for these environments are determined empirically, as shown in Table 4.5.

4.7 Key Insights: Frequent Memory Interactions

Once more, we take a closer look at the finite environments and ascertain that frequent memory interactions are demanded.

Mortar Mayhem comprises two distinct tasks: a clue task and an act task. During the clue task, there are as many write interactions as there are commands to be executed. While executing one command in the act task, the agent needs to recall the currently requested command. To perceive its position within the command sequence, the agent must memorize additional information. For example, the agent could count the executed commands to infer the next one. Thus, the agent needs to frequently alternate between reading from and writing to its memory. This process is illustrated in Figure 4.7.

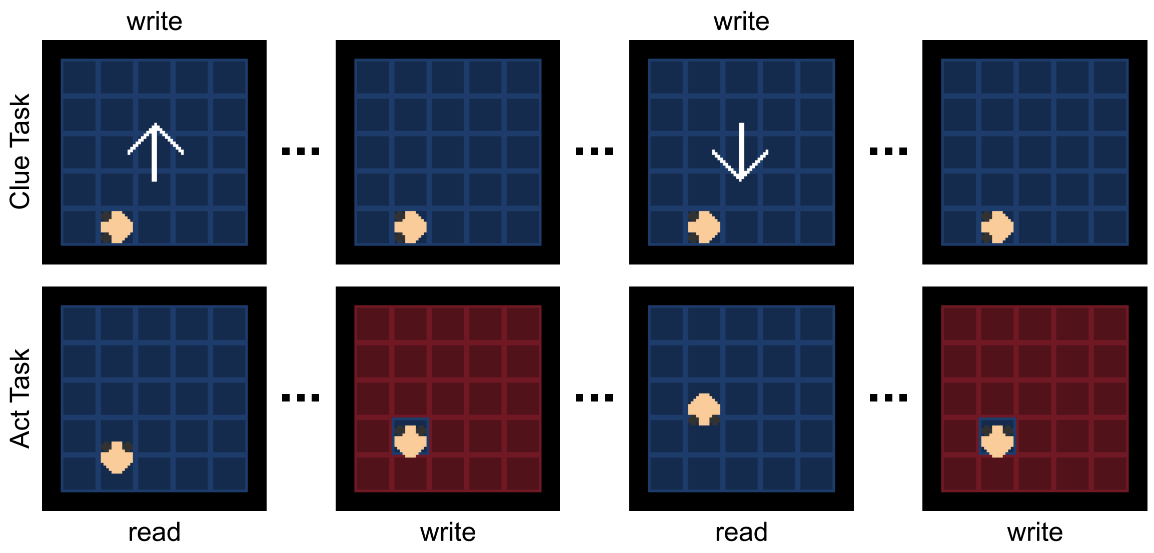


Figure 4.7: Within the clue task of Mortar Mayhem, the number of required write events matches the number of commands to execute. During the act task, the agent must alternate between reading and writing memory interactions.

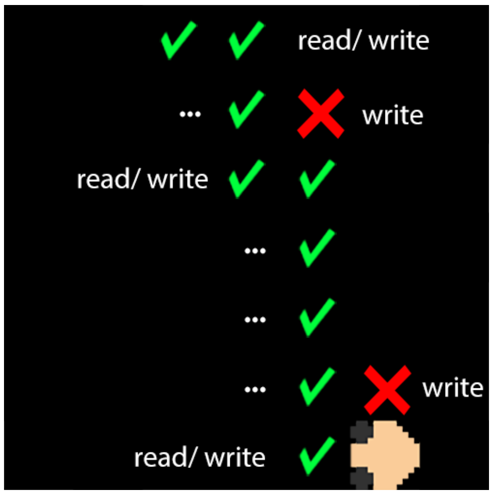


Figure 4.8: In Mystery Path, green ticks represent valid path positions, written to memory once. However, after falling off, this data, alongside fall-off locations, must be frequently accessed.

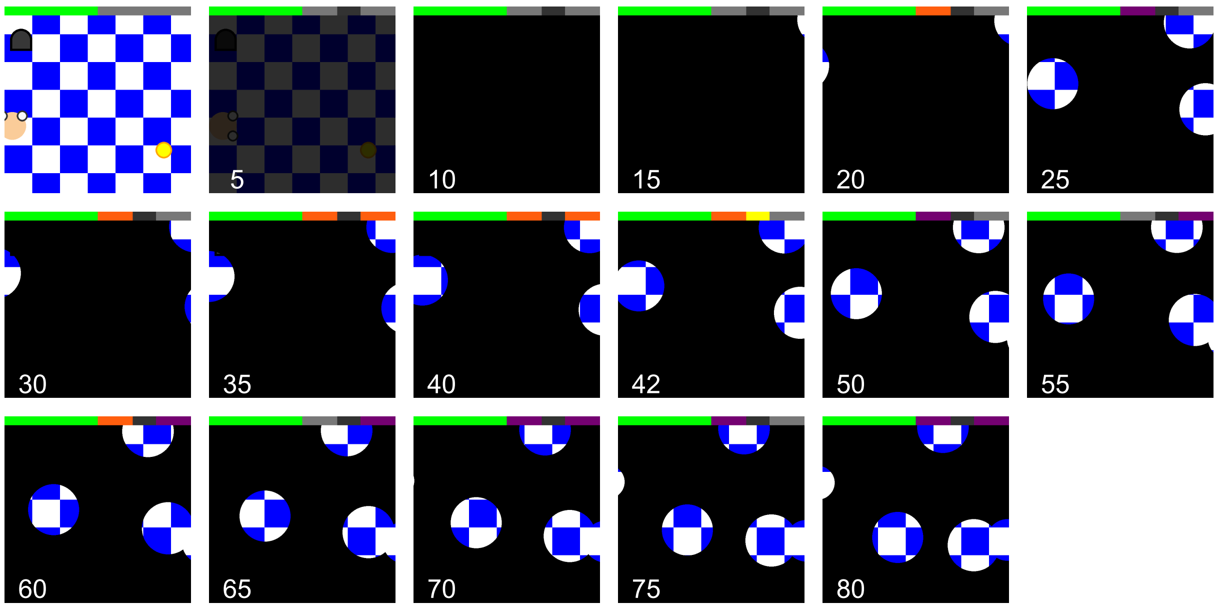


Figure 4.9: Episode visualization in Searing Spotlights. Annotations within the agent's observations detail the corresponding time steps. Within the initial 6 frames, the light gradually dims. During this transition, the agent commits to memory its position, as well as that of the coin and exit. Once obscured, the agent interacts continuously with its memory to store and retrieve prior actions, deducing its current location. The coin's collection, signified by the yellow rectangle in frame 42's top bar, requires memorization. During the remaining portion of the episode, the exit's location needs to be recalled.

Unlike Mortar Mayhem's static episodes, those in Mystery Path are dynamic. Given the procedural nature of the path — varying in length and complexity — the agent may fall off at different points or with different frequencies. As shown in Figure 4.8, each advancement on the hidden path necessitates a single memory write. However, after deviating from the path, the agent has to document the location and repeatedly access valid path positions and fall-off locations upon restarting.

In Searing Spotlights, the agent commits the following data to memory:

- Last visible positions of itself, the coin, and the exit.
- Actions taken since its last visibility.
- Coin collection events.

While singular events like positions and coin collections require one-time memory entries, previous actions demand ongoing storage and retrieval. Figure 4.9 offers a visual breakdown of an episode within this environment.

Chapter 5

Memory-based Deep Reinforcement Learning

We employ the widely-recognized DRL algorithm, Proximal Policy Optimization (PPO) (Schulman et al., 2017), which stands out as a model-free[1] and actor-critic method. The simplicity and potency of PPO have led to its success in varied domains ranging from video games, as seen in DotA 2 (Berner et al., 2019), to real-world robotic applications, such as dexterous in-hand manipulation (Andrychowicz et al., 2020), and even in natural language processing with large language models as ChatGPT (OpenAI, 2022).

However, PPO lacks a built-in mechanism to enable the ability of memory. To overcome this limitation, either RNNs or attention mechanisms (i.e. transformers) are potential solutions. Prior works have extensively demonstrated the effectiveness of leveraging RNNs, such as LSTM (Hochreiter and Schmidhuber, 1997) and GRU (Cho et al., 2014), within the realm of DRL (Mnih et al., 2016; Espeholt et al., 2018; Huang et al., 2022a). This also applies to previous studies employing transformers, as Gated Transformer-XL (GTrXL) (Parisotto et al., 2020) and Hierarchical Chunk Attention Mechanism (HCAM) (Lampinen et al., 2021). Adopting these memory mechanisms is not trivially plug-and-play. They entail an increase in implementation complexity due to gathering more data, processing sequential data, and extending model interfaces. This may lead to a higher risk of errors and an undesired memory overhead in terms of hardware memory usage. To make these baselines more accessible to the community, we contribute two easy-to-follow baseline implementations of GRU[2] and TrXL[3] (Dai et al., 2019) in PPO. It is worth noting that there is a dearth of readily available and accessible transformer-based DRL implementations (Sections 3.1.2 and 3.4). Therefore, our realization of TrXL plus PPO is especially valuable in this context.

The subsequent sections detail the actor-critic model architecture, discussing its utilization of a multi-discrete policy, an auxiliary observation reconstruction loss, and an optional ground truth estimation target. We then focus on the GRU PPO and TrXL PPO baselines, culminating in a brief summary.

[1]References to a model pertain to the agent's neural network architecture. "Model-free" denotes that the agent does not use a transition model for future predictions.
[2]GRU PPO Baseline: `https://github.com/MarcoMeter/recurrent-ppo-truncated-bptt`
[3]TrXL PPO Baseline: `https://github.com/MarcoMeter/episodic-transformer-memory-ppo`

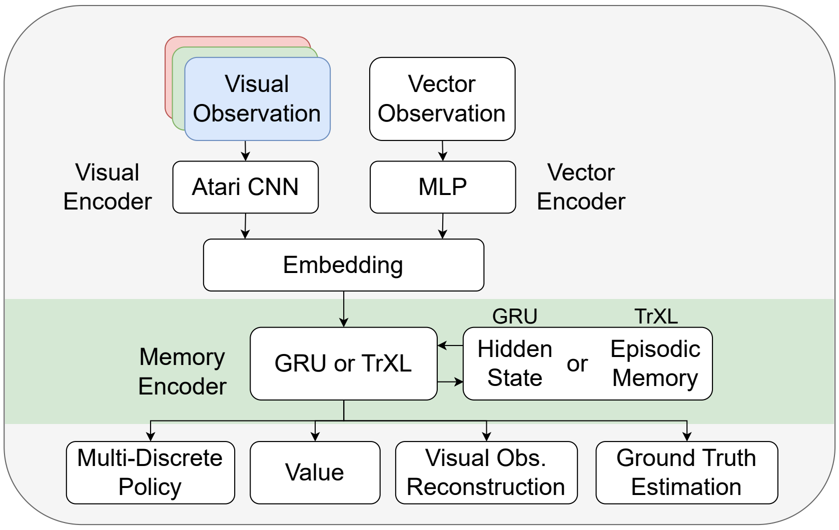


Figure 5.1: Overview of the actor-critic model architecture. The model features observation encoders and a memory encoder (green). The encoders' parameters are shared among several heads dedicated to the policy, state-value function, observation reconstruction, and ground truth estimation.

5.1 Model Architecture and Loss Composition

Figure 5.1 presents a broad overview of the actor-critic model architecture. The encoding process begins with visual observations, and optionally, vector observations (i.e. game state information). To encode visual observations (i.e. pixel observations) of shape $84 \times 84 \times 3$, we employ the Atari CNN, a convolutional neural network that adheres to the standard topology originally developed for solving Atari games (Mnih et al., 2015). The vector observation is encoded by an MLP. The outputs of these encoders are concatenated and subsequently embedded into the input space of the memory encoder, which is based on either GRU or TrXL. Once the data is propagated through the memory encoder, it is further processed by each individual head that may contain further fully connected hidden layers (i.e. MLPs). The policy head (actor) samples multi-discrete actions as required by Memory Gym, while the value head (critic) approximates the state-value. Optionally, there is a head dedicated to reconstructing the original visual observation and another one to estimate ground truth information provided as labels by the environment. Before presenting the final loss composition with respect to memory, the multi-discrete policy and the two optional auxiliary optimization targets, are detailed.

5.1.1 Multi-Discrete Action Policy

When trained on Memory Gym, an agent operates within a multi-discrete action space, as detailed in Section 4.2. The conceptual definition of an agent's actions remains abstract in chapter 2. In practical scenarios, an agent's set of actions can be discrete, continuous, or a hybrid of the two.

Owing to the specifications of Memory Gym, we first address the discrete action space and then progress to the multi-discrete paradigm. Let $\mathcal{A}$ be the action space of an agent. If this space consists of n distinct actions, then it can be characterized as a discrete action space:

$$\mathcal{A} = \{a_1, a_2, \ldots, a_n\} \tag{5.1}$$

Here, a_i denotes a unique action in the action space for all $1 \leq i \leq n$. Transitioning to the multi-discrete action space, consider $\mathcal{A}$ to encompass multiple discrete dimensions. This multi-dimensional nature can be expressed as union, while there are k dimensions:

$$\mathcal{A} = \bigcup_{j=1}^{k} \mathcal{A}_j \tag{5.2}$$

Each $\mathcal{A}_j$ signifies a discrete action space:

$$\mathcal{A}_j = \{a_{j1}, a_{j2}, \ldots, a_{jn_j}\} \tag{5.3}$$

Here, a_{ji} is a distinct action in the j^{th} discrete action space, with n_j being the count of unique actions within that dimension. In this multi-discrete setup, an agent's action is a vector $\mathbf{a} = [a_{1i}, a_{2i}, \ldots, a_{ki}]$, where each a_{ji} is sourced from its corresponding $\mathcal{A}_j$.

The primary benefit of employing a multi-discrete action space is the ability of an agent to simultaneously perform multiple actions. Take a keyboard as an example; several keys can be activated concurrently. While such a multi-action scenario can be translated into a single discrete action space, it mandates enumerating all possible action combinations. For instance, in the context of Memory Gym's action space with two discrete dimensions, each of size three, representing all combinations within a single discrete action space results in a total of 9 actions. This expansion complicates the credit assignment problem (Section 2.1.11) since the agent must now consider 9 actions instead of the initial 6.

To conclude this section, it is imperative to modify the policy loss formula, referenced as Equation 2.41 in Section 2.2.7. Each dimension in the multi-discrete action space is regarded as a distinct policy head. Consequently, the loss for every dimension, where only one action a_{ji} is selected, is determined and subsequently averaged:

$$L_t^C(\theta) = \frac{1}{k} \sum_{j=1}^{k} L_{t,a_{ji}}^C(\theta) \tag{5.4}$$

5.1.2 Observation Reconstruction

Utilizing self-supervised learning principles to enhance memory-based DRL agents with an observation reconstruction head has been shown to be effective (Hill et al., 2021; Lampinen et al., 2021). This technique is grounded in the autoencoder concept (Goodfellow et al., 2016, p. 499), where an input is processed through an encoder and then a decoder.

The resulting output should ideally match the initial input. A common application of autoencoders involves using the encoder for data compression and the decoder for decompression.

Within the context of DRL, the strategy entails attaching a decoder to reconstruct the initial, in our case, visual observation. Introducing this supplementary target amplifies the learning signal, aiming to improve the learning of suitable latent representations within the encoder. In practice, the latent output from the memory encoder is channeled through a transposed Atari CNN to reconstruct the original visual observation. During the development phase, attempts were made to connect the reconstruction head directly to the visual encoder; however, this approach resulted in low performances.

The parameters θ for both the encoders and the observation reconstruction head are optimized using the binary cross-entropy loss (Hill et al., 2021):

$$L^R(\hat{o}, o, \theta) = -\frac{1}{T}\sum_{t=0}^{T}\big(o_t \log(\hat{o}_t) + (1 - o_t)\log(1 - \hat{o}_t)\big) \tag{5.5}$$

In this equation, o_t represents the target at time t, which is the observation fed to the agent's model. Conversely, $\hat{o}_t$ refers to the predicted output from the observation reconstruction head.

Furthermore, observation reconstruction serves as a diagnostic tool. By solely optimizing the parameters of this head, one can assess whether the encoders have acquired adequate representations to enable a successful reconstruction. This head can either be trained concurrently with the agent or post agent-training.

5.1.3 Ground Truth Estimation

The ground truth estimation offers another diagnostic tool. This method relies on ground truth data from the environment, data that the agent does not observe. For a clearer illustration, Baker et al. (2020) trained memory-based agents in a multi-agent environment called hide and seek. After training, the agents were tested on various tasks. One such task challenges the agent to count boxes. The agent remains stationary, observing boxes moved in and out of its view field. Ultimately, the agent's estimation head processes the output from the agent's memory and tries to determine the exact number of boxes. This head's parameters ϕ are refined using the MSE (Section 2.2.2). The environment provides the labels, which serve as the ground truth.

We define the ground truth estimation loss as:

$$L^Y(\hat{y}, y, \phi) = \frac{1}{T}\sum_{t=0}^{T}(\hat{y}_t - y_t)^2 \tag{5.6}$$

Here, $\hat{y}_t$ represents the estimated value, y_t is the label, and T denotes the trajectory's length. Unlike the hide and seek example, we ask the agent to estimate the provided ground truth at every step.

Training just the ground truth estimation head is a valuable diagnostic measure in itself.

However, another diagnostic strategy involves allowing gradients to update the encoders' parameters θ as well. Even though using ground truth during the agent's training is now considered illegal, it can still offer valuable insights. As will be demonstrated in Section 7.1.2, the heightened effectiveness of an agent trained with ground truth estimation suggests that an agent without this feedback might not necessarily be lacking memory capacity.

5.1.4 Loss Composition for Memory Incorporation

Incorporating memory within optimization necessitates a revision of multiple loss definitions. Each optimization target discussed previously, as well as the value function (given by Equation 2.43), not only depends on the current observation o_t but also on the entire observation history. This history undergoes processing by the memory encoder, yielding a hidden state h_t.

The vector h_t serves as a context, encapsulating vital information from the observation history to restore the Markov property. Instead of repetitively adjusting each loss function associated with PPO, let us introduce a subtle simplification: any instance of the state s_t at a given time t is replaced with the combination of the current observation o_t and hidden state h_t.

The comprehensive loss for training a memory-based PPO agent can thus be represented as:

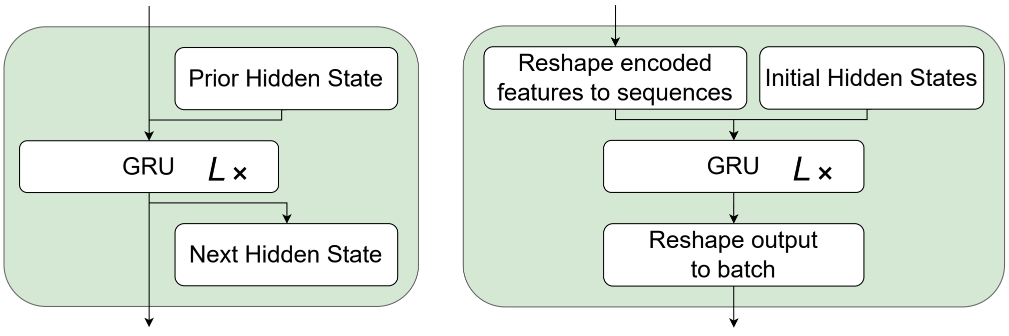
$$L_t^{C+V+H+R+Y}(\theta) = \mathbb{E}_t\left[L_t^C(\theta) + c_1 L_t^V(\theta) - c_2 H_t[\pi_\theta](o_t, h_t) + c_3 L_t^R(\theta) + c_4 L_t^Y(\theta)\right] \tag{5.7}$$

In the above equation, c_1 to c_4 are hyperparameters utilized for scaling their respective targets. The policy loss remains unscaled.

5.2 Gated Recurrent Unit Baseline

As detailed in Section 2.3, RNNs process sequential data iteratively by propagating information through recurrent connections. This characteristic is harnessed by various architectures as GRU and LSTM, both of which are supported in our baseline implementation. Also, our baseline also accommodates truncated back-propagation through time, vital for managing long sequences.

This section delves into the nuances of integrating a GRU cell into the agent, primarily to equip it with memory capabilities. This choice for GRU over LSTM was not arbitrary. Our preference for GRU in our major experiments stemmed from two pivotal reasons. Firstly, the GRU design is thinner in terms of GPU memory consumption because it exclusively depends on its hidden state, foregoing the additional cell state characteristic to LSTM. Moreover, GRU offers a computational advantage; it is less taxing, as demonstrated by Lambrechts et al. (2022) and Morad et al. (2023). This is also apparent in the results of our experiments on Memory Gym's finite environments (Section 6.2.3).



(a) Inference (b) Optimization

Figure 5.2: An overview of the two distinct forward passes concerning inference and optimization when incorporating an RNN, such as GRU (stacked L times), as a memory encoder. During data sampling (a), the agent interacts with its environment step by step, updating the prior hidden state based on the encoded features of the current observation. In the optimization phase (b), observation encodings and formerly collected hidden states are divided into sequences. Each sequence is then processed by unfolding the recurrent cell starting from its corresponding initial hidden state. The output is reshaped to its original batch dimensions.

Venturing deeper into the GRU baseline, we spotlight three major components that are pivotal for a working and efficient implementation

- The forward pass differs between training data sampling and the optimization phase.
- To accelerate the forward pass, training data — organized in sequences of varying lengths — is padded, albeit at the cost of increased GPU memory overhead.
- It is then crucial to mask the introduced paddings during the loss computation.

Lastly, we briefly touch on significant implementation bugs that have hampered progress for a long time.

5.2.1 Varied Forward Passes in Data Sampling and Optimization

The model's forward pass changes based on whether it is for inference (sampling training data) or optimization, as illustrated in Figure 5.2. For data sampling:

- The recurrent cell starts with an initial hidden state of zeros.
- As it interacts with the environment step by step, the cell produces a new hidden state, influenced by the previous one and the current observation.

- All produced hidden states during this process are saved for future optimization.
- If the environment resets because of termination, the hidden state goes back to zeros.
- If an episode has not finished by the end of a trajectory, the next sampling phase continues from where it stopped. It uses the former hidden state and the environment's present state.
- During this stage, no gradients for optimization are computed.

Concerning optimization :

- The forward pass works on a data batch, not individual agent-environment interactions as during inference.
- When propagating through the non-recurrent layers, it is more efficient to use the entire batch instead of a batch of sequences.
- Once data reaches the recurrent cell, it should be given as a batch of sequences. The recurrent cell then processes it and unfolds itself internally.
- Subsequently, the data is reshaped to its original batch dimensions.

During the forward pass of this phase, it is crucial that all sequences are of the same length. Due to the possible variability in episode lengths in RL tasks, sequences often need padding. This process, and how the data is split into sequences, will be described subsequently.

5.2.2 Splitting Training Data into Sequences

Before optimization, the sampled observations need to be split into sequences. As illustrated by Figure 5.3, this data is first divided into episodes. If desired, full episodes, most likely including truncated ones, can be considered as the final sequences. Therefore, the maximum sequence length is determined by the longest episode within the collected trajectories. If a fixed sequence length is set before training, the episodes are further split into sequences of that length. Finally, sequences shorter than the designated or maximum length are zero-padded to optimize computational efficiency, while considering some computer memory overhead. At this point, all sequences have the same length and can be efficiently consumed during optimization.

This process is also executed for the sampled hidden states. While there is no reason for inputting sequences of hidden states, it simplifies the process of selecting the initial hidden states for every processed sequence. Because of this, sequences that do not start with the first time step, begin with their corresponding intermediary hidden state, enabling truncated back-propagation through time (Section 2.3.1).

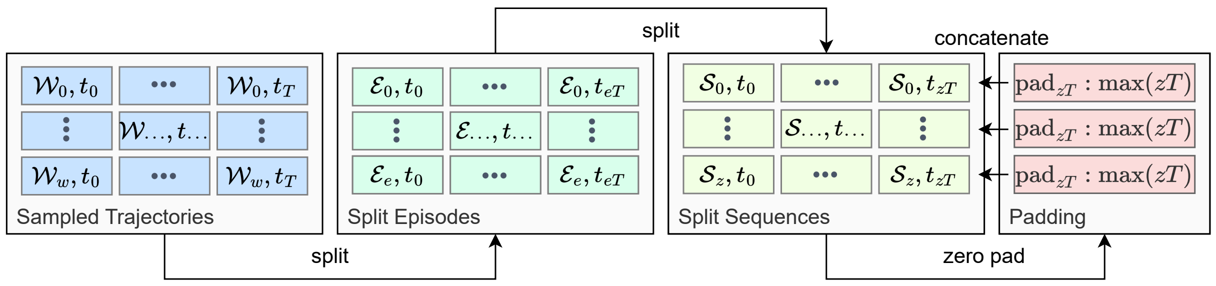


Figure 5.3: Arranging sampled training data into zero-padded sequences of fixed length. The data processing starts out by sampling trajectories across w workers for T steps. Next, e episodes of varying length eT are extracted from the trajectories. Those can be further split into z sequences of varying length zT. At last, zero padding is used to retrieve sequences of fixed length $\max(zT)$.

5.2.3 Loss Masking to Remove Paddings

As a consequence of using zero padding, the padded values need to be excluded during loss computation. This is accomplished by leveraging a mask, where L^{mask} is the average over all losses not affected by the padding.

$$L^{mask}(\theta) = \frac{\sum_t^T \left[mask_t \times L_t^{C+V+H+R+Y}(\theta)\right]}{\sum_t^T [mask_t]} \tag{5.8}$$

$$\text{with } mask_t = \begin{cases} 0 & \text{where padding is used} \\ 1 & \text{where no padding is used} \end{cases}$$

5.2.4 Challenges in Implementing DRL with Recurrent Layers

DRL algorithms are generally prone to bugs that are difficult to trace. At a glance, integrating a recurrent layer into a model seems simple. However, the intricacies of doing so go beyond mere architectural adjustments. The entire training algorithm setup becomes more nuanced and complex. In our journey to establish the GRU baseline, numerous bugs arose, stalling our progress for extensive periods. Three particularly vexing issues emerged due to nuances in the PyTorch library:

- When unintentionally providing "None" as the hidden state to a recurrent layer, PyTorch defaults the network's hidden states to zero, leading to exploding gradients.

- PyTorch's recurrent layer assumes the sequence length to be the input data's primary dimension. Careless reshaping can disrupt sequence data, misplacing items among unrelated sequences. This can be avoided by activating the "batch first"[4] setting during the recurrent layer's forward operation.

- It is vital to remember that outputs from both LSTM and GRU cells are already activated. Applying another activation, as Rectified Linear Unit (ReLU) (Fukushima, 1969), may degrade performance.

In light of these circumstances, we recognize the importance of offering clear, well-documented, and user-friendly baseline implementations, and have made efforts in this section and within our GitHub repository to provide just that.

5.3 Transformer-XL Baseline

When leveraging a GRU cell as memory mechanism, the essential components are the features (i.e. activations) of the encoded observation and the prior hidden state. On the other hand, a transformer encoder adopts a different strategy. Rather than sequentially maintaining and updating a single hidden state, it simultaneously processes the entire history of observations. At a glance, this might resemble the strategy of simply stacking frames. However, due to its attention mechanism, the transformer can prioritize and focus solely on the relevant parts of the observation history.

Our attention-based baseline is conceptually influenced by the ideas presented in TrXL (Dai et al., 2019), GTrXL (Parisotto et al., 2020), and HCAM (Lampinen et al., 2021). While these studies lack exhaustive explanations and often do not provide functional source codes, our aim is to bridge these gaps with a transparent implementation and detailed insights into the concept.

The primary pieces of our Transformer-XL baseline include:

- Addressing a sequence-to-one approach, a common scenario in RL tasks.
- Incorporating an episodic memory to store prior inputs (also referred to as hidden states) for the TrXL blocks, thereby supporting the segment-level recurrence characteristic of TrXL.
- Utilizing a sliding window, which incrementally moves across the episodic memory in tandem with the agent's progression within its environment. This ensures that the sequence length of prior hidden states remains manageable, considering computational constraints.
- Utilizing positional encoding to embed sequence order information. Our implementation supports absolute, learned, and relative approaches.

This section will further delve into the episodic memory and its associated sliding window mechanism, followed by an in-depth examination of the transformer block.

[4] `https://pytorch.org/docs/stable/generated/torch.nn.GRU.html` Accessed: 2023-09-18

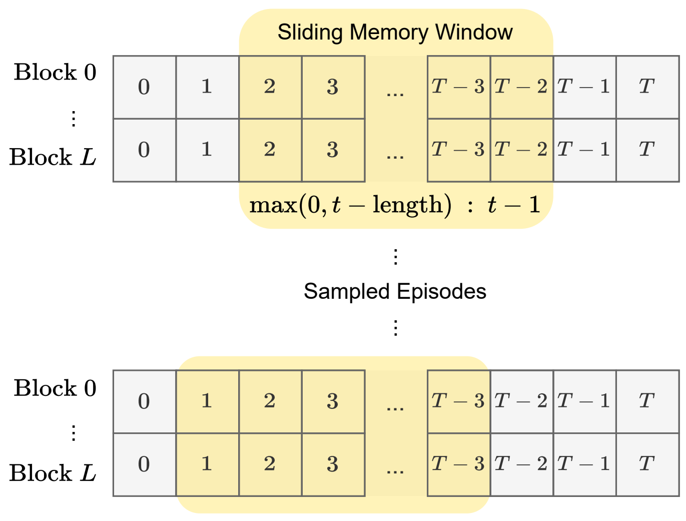


Figure 5.4: Illustration of the episodic memory which retains past inputs across all Transformer-XL blocks for every agent's step in an episode. A sliding memory window of a predetermined length helps manage computational demands. The segment-level recurrence methodology ensures that events outside the window's boundaries remain accessible. The TrXL blocks receive input sequences via this sliding memory window. T represents the episode length and t denotes the current time step.

5.3.1 Episodic Sliding Memory Window

The segment-level recurrence demands the storage and upkeep of a complete episode of past hidden states. This becomes particularly challenging within the context of DRL. In scenarios where episodes might be truncated during a PPO iteration, subsequent iterations would still necessitate the initial fragment of hidden states from the episode. Hence, we employ an episodic memory to consistently store entire episodes across these iterations.

As the agent engages with its environment, the episodic memory accumulates data. The transformer processes the series of hidden states from the ongoing episode, recalling historical details to make informed decisions. Given computational constraints, especially in tasks that could continue indefinitely, we utilize a sliding memory window strategy. In this, the input sequence is extracted from a window of a fixed length (Figure 5.4).

Another challenge emerges during the optimization phase. Storing memory windows corresponding to every individual time step and worker becomes computationally burdensome. Given that each window's position often overlaps with its predecessor, it results in significant data redundancy. To address this, each worker retains its series of time steps based on the momentary position of the memory window. Retaining the time steps also aids in pinpointing the appropriate positional encodings for each sequence element. To prevent the agent from accessing future hidden states — particularly when the current time step is shorter than the window's length — a mask is employed. This ensures that such data is excluded during multi-head attention.

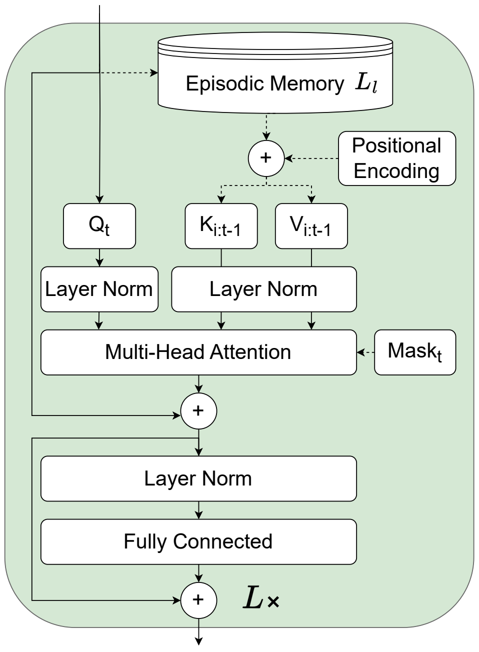


Figure 5.5: Detailed visualization of our Transformer-XL block architecture that is stacked L times. The dashed lines indicate that there is no flow of gradients into the episodic memory, the absolute positional encoding, and the mask.

5.3.2 Detailed Transformer-XL Block

The architecture of our TrXL block is visualized in Figure 5.5. At each time step t, the block receives a single input that is based on the current time step. This input is added to the episodic memory and serves as the query Q_t during the execution of MHA. Note that the query is a single vector rather than a sequence. The keys K and values V used in MHA are derived from the same data based on self-attention. Specifically, we retrieve K and V by slicing the episodic memory using the bounding indices $i = \max(0, t - \text{window length})$ and $t - 1$.

To ensure the positional information of K and V remains coherent, we add an absolute or learned positional encoding, following the approach from Vaswani et al. (2017). We use the absolute variant as a default as we found the learned one to be effective, but less sample efficient. The underlying positional encoding matrix is scaled to the maximum episode length and not to the length of the window. Therefore, every sequence element of an episode is associate with its distinct position. In the case of endless episodes, the length of the positional encoding is fixed to 2048.

To restrict attention to time steps up to the current time step t, we employ a strictly lower triangular matrix as a mask in MHA. This mask is only necessary when t is smaller than the length of the sliding memory window. The remaining configuration of the block adheres to the findings of Parisotto et al. (2020), which suggest improved performance with pre layer normalization and the identity map reordering (Figure 2.11 (b)). We encountered vanishing gradient issues when leveraging post layer normalization. Although our implementation supports the gating mechanism of GTrXL, it resulted in either lower or equivalent performance while being computationally more expensive (Section 6.2.3).

5.3.3 GPU Memory Overhead Challenges

The episodic transformer memory implementation poses significant memory overhead challenges. For instance, when extracting memory windows for optimization, take into account a batch size of 16384, 3 TrXL blocks of dimension 384, and a window length of 256. This configuration, solely for the hidden states, demands 18GB of GPU memory, even when the NVIDIA A100 GPU offers a total of 40GB. In contrast, a batch of visual observations with dimensions $84 \times 84 \times 3$ only occupies 1.29GB. The challenge amplifies when slicing memory windows. Operations on tensors can lead to instantiating a new tensor, where additional memory is allocated. For example, slicing tensors with masks introduce this overhead. Although unused memory is freed once the operation completes, the required GPU memory may already be surpassed during the operation itself.

Another memory overhead concern emerges during data sampling. Before initiating the sampling process, it is uncertain how many episodes of varying lengths will be gathered. Consequently, whenever a new episode commences, a tensor as long as the maximum episode length is setup. This approach can result in considerable and inefficient padding if numerous short episodes are sampled. Such complications make it challenging to upscale the transformer. An alternative is to use CPU memory as well, but this comes at the cost of expensive IO time.

5.4 Key Insights: Comparing both Baselines

To summarizes this chapter, we examine the memory details of the proposed GRU and TrXL baselines and recite the actor-critic model architecture.

Both baselines require careful processing of sequential data, necessitate padding, and incorporate masking. The primary differentiation is their approach to handling sequential data. The GRU baseline processes it incrementally, whereas the TrXL baseline consumes the entire sequence in one go. Notably, retaining the initial hidden states of a recurrent cell is more memory-efficient than preserving all hidden states across the entire transformer blocks for a full episode. Consequently, managing the computational memory overhead of the TrXL baseline posed a greater challenge in our implementation.

Revisiting the actor-critic model architecture, visual observations are processed by the Atari CNN, while optionally vector observations are fed to a fully connected layer. The various heads of the model connect to the outputs of the aforementioned memory encoders. One head is explicitly designed to accommodate the multi-discrete action space. Another is tasked with estimating the state value, serving the critic's function. An additional avenue to enhance the learning signal and foster better encoder representations is to utilize a visual observation reconstruction head. For a deeper diagnostic perspective, it is also an option to integrate a ground truth estimation target.

Chapter 6

Experimental Analysis

To assess the efficacy of the just introduced GRU and TrXL baselines, both backed by Proximal Policy Optimization (Schulman et al., 2017), we conduct comprehensive experiments across all Memory Gym environments. These experiments can be reproduced by leveraging our publicly available research framework[1].

In this chapter, we begin by delineating our experimental protocol, including hyperparameter tuning, utilized statistical tools, and the approach for evaluating generalization. Subsequent to this, we reveal the outcomes from the finite environments, emphasizing the memory dependency of all environments. The TrXL agent proves to be more capable in Mortar Mayhem and sample efficient in Mystery Path, while the GRU agent is more efficient in Searing Spotlights. Preliminary assessments of GTrXL and LSTM architectures yield modest, often inferior, results. A focal point is then the unexpected performance in the endless environments, where the GRU agent distinctly surpasses TrXL. Additionally, a comparison of wall-time costs indicates that GRU is more cost-effective. Prior to summarizing the pivotal findings, we delve into the peculiarities of agent behaviors post training.

6.1 Experimental Protocol to Assess Generalization

To ensure reliable and reproducible benchmark results, we detail the protocols used for experimentation and evaluation in the sections that follow. First, we present the hyperparameters and their tuning process. Next, we delve into the specific architecture of the agent's ANN. Given the significant costs associated with training DRL agents, we can only conduct a limited number of experiment repetitions. To address the statistical uncertainty arising from this limitation, statistical methods recommended by Agarwal et al. (2021) are utilized. Following this, we detail the score metrics of the environments and describe the approach to assessing generalization.

[1]DRL Research Framework: `https://github.com/MarcoMeter/neroRL`

Table 6.1: Hyperparameters and architectural details used in our experiments. The "Final" column denotes the selected hyperparameter values, while the "Search Space" column represents additional discrete choices explored during the tuning process. The last column details the old hyperparameters of our previous study (Pleines et al., 2023). For this column alone, empty cells indicate that the values align with those in the "Final" column. Xavier (Glorot and Bengio, 2010), Kaiming (He et al., 2015), and T-Fixup (Huang et al., 2020) are weight initialization approaches. Most of the parameters of the other ANN are initialized orthogonally. The AdamW optimizer is contributed by Loshchilov and Hutter (2019).

Hyperparameter	**Final**	**Search Space**			**Old**
Training Seeds (Levels)	100,000				
Number of Workers	32				
Worker Steps	512				
Batch Size	16384				
Disocunt Factor Gamma	0.995				0.99
GAE Lamda	0.95				
Optimizer	AdamW				
Epochs	3	2	4		
Number of Mini Batches	8	4			
Advantage Normalization	No	Batch	MB		MB
Clip Range Epsilon	0.1	0.2	0.3		0.2
Value Loss Coefficient	0.5	0.25			0.25
Initial Learning Rate	2.75e-4	2.0e-4	3.0e-4	3.5e-4	3.0e-4
Final Learning Rate	1.0e-5				1.0e-4
Initial Entropy Coefficient	1.0e-4	1.0e-3	1.0e-2		
Final Entropy Coefficient	1.0e-6				1.0e-5
Reconstruction Loss Coef.	0.1	0.5	1.0		n/a
Ground Truth Est. Coef.	0.1				n/a
Maximum Gradient Norm	0.25	0.35	0.5	1.0	0.5
Recurrent Neural Network					
Number of Recurrent Layers	1	2			
Use Embedding Layer	Yes				No
Layer Type	GRU	LSTM			
Residual	False	True			
Sequence Length	512 or max				
Hidden State Size	512	256	384		
Transformer-XL					
Number of TrXL Blocks	3	2	4		
Block Dimension	384	256	512		
Block Weight Initialization	Xavier	Orthogonal	Kaiming	T-Fixup	
Positional Encoding	Absolute	None	Learned		
Number of Attention Heads	4	8			
Memory Window Length	256 or max				

Table 6.2: Detailed overview of the model architecture: The Atari CNN (Mnih et al., 2015) features 3 convolutional layers to process visual observations, resulting in a vector of dimension 3136 (64×7×74). For training specifically on Mortar Mayhem's act task, the 10 commands are one-hot encoded, resulting in a vector observation of size 180. In experiments proving memory dependence, a memory-free agent is trained with the help of positional encoding, leading to an additional vector observation of size 16. When both types of observations are used, they are concatenated and embedded to match the input dimensions of the designated memory encoder: 384 for TrXL and 512 for GRU. The memory encoder is completed by appending a fully connected layer. Every model head has its own fully connected layer to begin with. All fully connected layers are activated using ReLU (Fukushima, 1969). Figure 5.1 depicts the model's flow.

Observation Encoder	**In Dim**	**Out Dim**	**Filters**	**Size**	**Stride**
Convolutional Layer 1	3×84×84	32×20×20	32	8	4
Convolutional Layer 2	32×20×20	64×9×9	64	4	2
Convolutional Layer 3	64×9×9	64×7×7	64	3	1
Vector Observation Mortar Mayhem Act		180			
Vector Observation Positional Encoding		16			
Concatenate Vector Observation		196			
Fully Connected	196	128			
Concatenate Observations		3264			
Observation Embedding Fully Connected	3264	384 / 512			
Memory Encoder	**In Dim**	**Out Dim**	**Layers**		
Gated Recurrent Unit	512	512	1		
Transformer-XL	384	384	3		
Fully Connected	384 / 512	512			
Multi-Discrete Policy	**In Dim**	**Out Dim**			
Fully Connected	512	512			
Discrete Policy Dimension 1	512	3			
Discrete Policy Dimension 2	512	3			
State-Value Head	**In Dim**	**Out Dim**			
Fully Connected	512	512			
State-Value Estimation	512	1			
Observation Reconstruction	**In Dim**	**Out Dim**			
Fully Connected	512	64×7×7			
Transposed Conv. Layers 1-3	64×7×7	3×84×84			
Ground Truth Head	**In Dim**	**Out Dim**			
Fully Connected	512	512			
Ground Truth Estimation	512	2			

6.1.1 Choice of Hyperparameters and Model Architecture

General Settings: The second column of Table 6.1 presents the hyperparameters used in forthcoming experiments unless otherwise noted.

- The learning rate and entropy coefficient linearly decay from their initial to final values during the first 10,000 PPO iterations, equivalent to 163,840,000 steps. After this, the final hyperparameter serves as a lower threshold.
- Sequence lengths and memory window lengths are based on the longest episode in finite environments. For endless environments, sequence lengths of 512 for GRU and 256 for TrXL are chosen.
- The batch size is computed by multiplying the number of environment workers with the steps each worker takes to gather data. On a system with 32 CPU cores and 40GB VRAM on an NVIDIA A100 GPU, we opted for a batch size of 16,384 samples. For larger models, either reducing the batch size or moving data to CPU memory is required, but both methods increase wall-time.

Hyperparameter Tuning Approach: We utilized a custom implementation based on the Optuna tuning framework (Akiba et al., 2019) to efficiently search for suitable hyperparameters. The goal was to identify a single set of hyperparameters offering robust performances across Memory Gym's environments for both GRU and TrXL baselines.

- Our search space was constrained by resources. Most tuning experiments centered on finite environments; thus, the hyperparameters may not be globally optimal.
- For each environment, dedicated tuning experiments were conducted based on Table 6.1. A single parameter from the search space was adjusted in each trial, resulting in at least 29 trials for TrXL and 24 for GRU. On average, each baseline and environment underwent 67 trials.
- Using Optuna's percentile pruner, only the top 25 percentile of trials were retained. Trials below this cutoff were pruned after a 2,000 PPO iteration warm-up phase.
- Once all single choices were tried, an expanded search allowed permutations using Optuna's Tree-structured Parzen Estimator (Bergstra et al., 2011), a Bayesian optimization technique. This method fits two Gaussian Mixture Models to hyperparameter values and chooses the value x maximizing the $l(x)/g(x)$ ratio.
- Individual trials were not repeated, but since tuning spanned multiple environments, these runs are considered as repetitions.

Architecture and Scale of the Agent's ANN: Table 6.2 outlines the dimensions of each layer within the agent's model. For a glimpse of the number of trainable parameters, consider training the GRU and TrXL baselines in Endless Mystery Path, excluding auxiliary heads for observation reconstruction and ground truth estimation. The GRU agent contains approximately 4.05 million trainable parameters, while the TrXL one has around 2.8 million. By expanding TrXL to GTrXL and therefore incorporating 6 GRU cells, the total trainable parameters increase to 9.4 million.

6.1.2 Statistical Tools for Reporting Sample Efficiency Scores

Before exploring the results of the contributed baselines on Memory Gym, it is essential to clarify the statistical tools in use. Given the inherent stochastic behavior of environments, agents, and optimization, multiple training runs are essential to get a reliable picture of the agents' performances. The feasibility of numerous repetitions is often constrained by the computational and time costs associated with a DRL algorithm and task. For example, benchmarking PPO was limited to only 3 repetitions (Schulman et al., 2017). In this study, each experiment is repeated 5 times, each with a distinct random seed. To address the challenges posed by statistical uncertainty, the guidelines set forth by Agarwal et al. (2021) are followed, employing their advocated point and interval estimates. The term "estimates" is emphasized here due to the variability stemming from the finite and low number of runs.

Point estimates refer to data aggregation methods, such as computing the mean, which can be skewed by outliers. Hence, the Interquartile Mean (IQM), which focuses on the middle 50% of data, is recommended as a more robust alternative (Agarwal et al., 2021).

Interval estimates provide a range within which the true performance value is believed to lie. A commonly used type is the Confidence Interval (CI). A 95% CI suggests that, if the experiment were to be repeated many times, the true performance value would fall within this interval 95% of the time. An actual score outside this 95% CI implies a rare event, with only a 5% likelihood of occurring by chance. For the underlying situation of limited samples, Agarwal et al. (2021) recommend stratified bootstrap confidence intervals. Bootstrapping involves resampling from the existing data with replacement (Efron, 1979). Stratification ensures that each run is adequately represented in the bootstrap samples, leading to more reliable and robust confidence intervals (Agarwal et al., 2021).

When evaluating the trained baselines, the IQM and CI, which is based on the stratified bootstrap method, are used. These metrics form the foundation for the upcoming sample efficiency curves, presented as line plots. The x-axis represents time in steps, and the y-axis displays the agent's aggregated scores across completed episodes. The bold line in the plots signifies the IQM, with the surrounding shaded region representing the CI.

6.1.3 Evaluating Generalization

During training, the agent encounters 100,000 procedurally generated levels. For assessing generalization, the performance is evaluated on 50 unseen levels post-training. Given the stochastic nature of the environment and the agent's policy, each of these levels is revisited three times. In the sample efficiency curves, each data point aggregates results from 5 training runs over 50 levels, with each level repeated three times, leading to a total of 750 episodes per data point.

The type of agent scores differ depending on the specific environment. For all Mortar Mayhem variants, the designated metric is the count of successfully executed commands. For finite versions of Mystery Path and Searing Spotlights, the success rate, indicating the completion of all task objectives, is used. Concerning the remaining endless tasks, Mystery Path employs the undiscounted return, while Searing Spotlights uses the total of gathered coins.

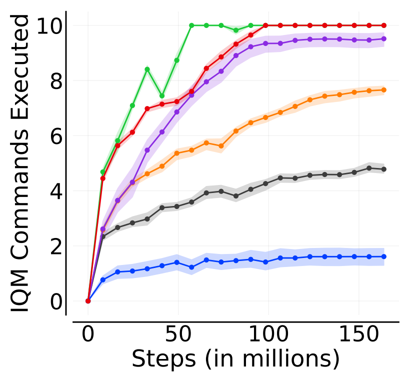
6.2 Results in Finite Environments

This section presents the outcomes from several experiments on training agents in Memory Gym's finite environments. Initially, the first set of experiments ensures that all environments require agents to effectively utilize memory. This understanding comes from contrasting the contributed baselines with minor agent architectures, such as frame stacking, in more basic environment versions. Subsequent comparisons between GRU and TrXL in the default finite tasks show that TrXL excels in Mortar Mayhem and Mystery Path but is less efficient in Searing Spotlights. At last, preliminary results emerge when employing GTrXL and LSTM in a plug-and-play manner, underscoring the importance of hyperparameter tuning and implementation verification.

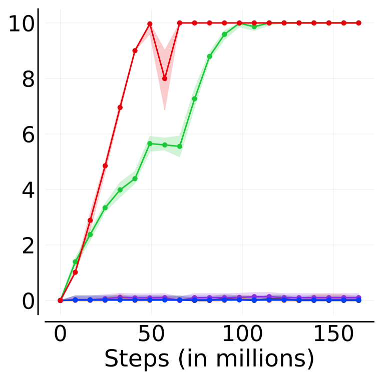
6.2.1 The Finite Tasks Depend on Agents Using Memory

Before assessing GRU and TrXL on the default challenges of the finite environments, the first set of experiments focuses on confirming that the agents trained on Memory Gym indeed require memory. To accomplish this, we train several minor baselines alongside GRU and TrXL on simplified versions of the environments. These baselines include PPO without memory, PPO with frame stacking (4 and 16 grayscale frames), and PPO with absolute positional encoding (Section 2.4.3), which is provided as vector observation. By observing the positional encoding, the agent is informed of the current time step, with the encoding dimension set to 16 and the length matching the maximum episode length.

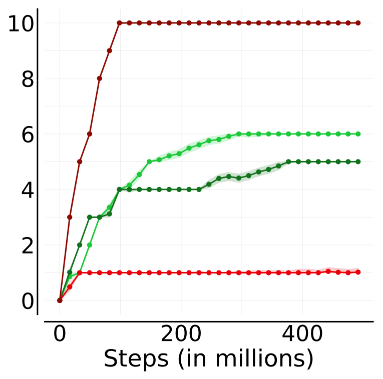
Mortar Mayhem Depends on Memory
Results on Mortar Mayhem (Figures 6.1 (a)- (c)) show the number of commands that were properly executed during one episode. As mentioned in Section 4.3.1, the agent is not required to memorize the command sequence in the act task as it is included in the agent's vector observation. Under these simpler conditions, PPO without memory is ineffective, while the frame stacking agents achieve nearly either 5 or 8 commands with a slight upward trend. Agents using GRU or TrXL are the most effective, completing all 10 commands, while those with relative positional encoding follow closely, completing 9.5 commands. Once the clue task is present (Figure 6.1 (b)), only GRU and TrXL prove to be effective.



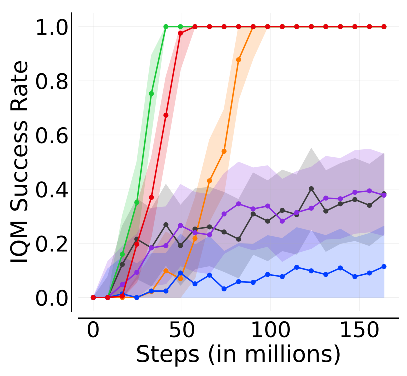
(a) Mortar Mayhem Act Grid



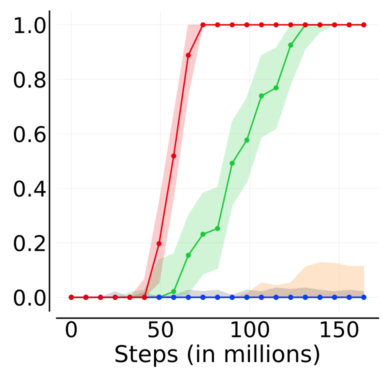
(b) Mortar Mayhem Grid



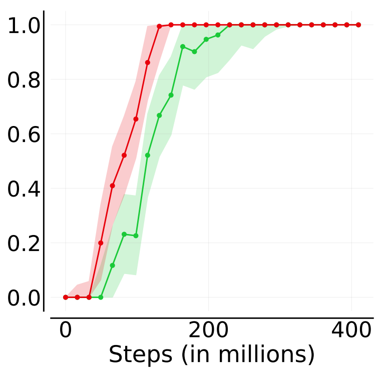
(c) Mortar Mayhem



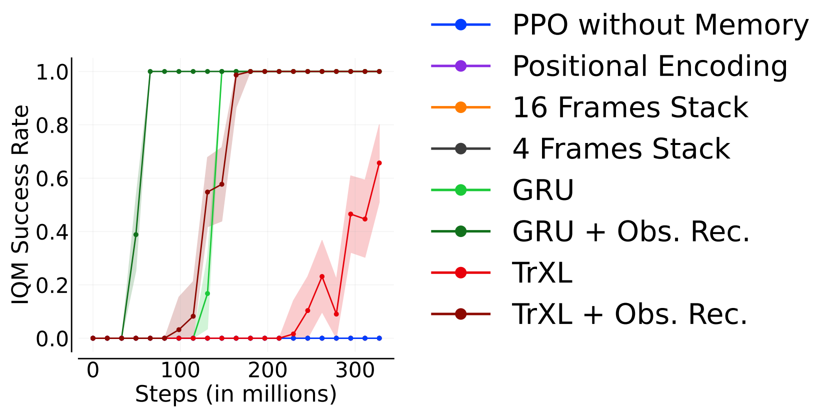
(d) Mystery Path Grid (Cues On)



(e) Mystery Path Grid



(f) Mystery Path



(g) Searing Spotlights

Figure 6.1: Performance comparison across instances of Memory Gym's finite environments. Mortar Mayhem Act Grid (a) provides commands as a fully observable vector, bypassing the clue task. Both (a) and Mortar Mayhem Grid (b) utilize grid-like locomotion. (c) represents the default Mortar Mayhem task. Mystery Path Grid operates on grid-like locomotion, with (a) rendering the goal and origin to the agent's observation. (b) and (c) hide these, while (c) is the default Mystery Path task. Searing Spotlights (g) is not varied, while all conducted runs leverage observation reconstruction (Obs. Rec.) in this environment.

Mystery Path Depends on Memory
We obtain a similar impression when training on Mystery Path Grid where the origin and goal are not perceivable by the agent (Figure 6.1 (e)). In this case, TrXL outperforms GRU in terms of sample efficiency, while both methods are effective. If the origin and the goal are visible, a horizon of 16 frames is sufficient to train an effective policy as shown by the frame stacking agent (Figure 6.1 (d)). Stacking 4 frames or leveraging positional encoding leads to an IQM success rate of 40% with a slight trend upward. The memory-less agent's success rate goes up to about 11%.

Searing Spotlights Depends on Memory
No success is proven by the minor baselines in Searing Spotlights as seen in Figure 6.1 (g). Only in Searing Spotlights, the minor baselines also utilize observation reconstruction.

6.2.2 Transformer-XL Outperforms Gated Recurrent Unit, But Not in Searing Spotlights

Based on the results, as seen in Figure 6.1, TrXL proves to be more

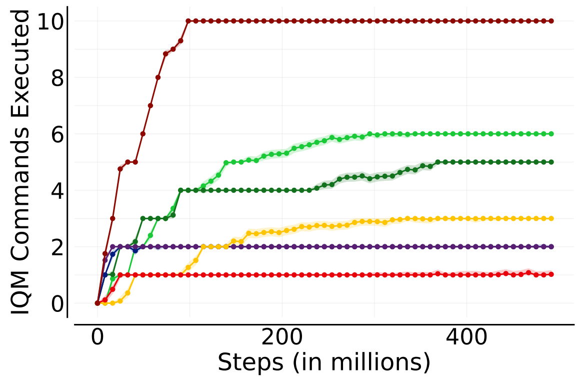
- effective in Mortar Mayhem (Figure 6.1 (c)),
- efficient in Mortar Mayhem Grid (Figure 6.1 (b)), and more
- efficient in Mystery Path (Figure 6.1 (f)) and its grid variant (Figure 6.1 (e)).

Concerning the GRU agent, it proves to be more efficient in

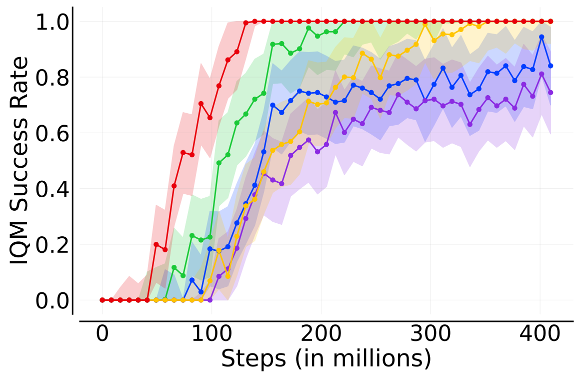
- Mortar Mayhem Act Grid (Figure 6.1 (a)),
- Mystery Path Grid (Figure 6.1 (e)), where the origin and goals are observable, and
- Searing Spotlights (Figure 6.1 (g)).

In Mortar Mayhem, the command list is displayed only temporarily, making the encoding of these rare events pivotal. Leveraging observation reconstruction plays a vital role in this context, proving indispensable for an effective TrXL agent. However, this auxiliary target hinders the GRU agent, decreasing its performance from 6 to 5 commands. Conversely, for Mystery Path, we opted not to train agents using observation reconstruction, as the always visible agent is the only entity that requires visual encoding.

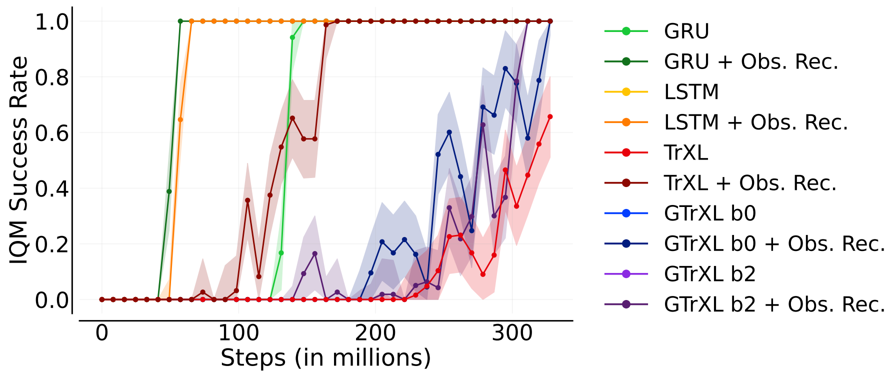
In summary, TrXL surpasses GRU in terms of effectiveness in Mortar Mayhem and efficiency in Mystery Path. Conversely, GRU leads in Searing Spotlights with respect to efficiency.



(a) Mortar Mayhem



(b) Mystery Path



(c) Searing Spotlights

Figure 6.2: Preliminary results for GTrXL and LSTM in the finite environments. Without comprehensive hyperparameter tuning, both GTrXL and LSTM perform less effectively in Mortar Mayhem and show reduced sample efficiency in Mystery Path and Searing Spotlights. The terms "b0" and "b2" denote the biases explored within the GTrXL context.

6.2.3 Preliminary Results with Gated TrXL and LSTM

Figure 6.2 presents extended results for the finite environments. We include preliminary results from agents utilizing GTrXL (Parisotto et al., 2020) and LSTM (Hochreiter and Schmidhuber, 1997). These models are benchmarked without thorough hyperparameter tuning. For GTrXL, biases of both 0 and 2 are tested, based on the suggestion by Parisotto et al. (2020) that a larger bias within the GRU cells might enhance learning. All additional agents in the Searing Spotlights environment employ the observation reconstruction loss (Obs. Rec.), while it is omitted in the Mystery Path environment.

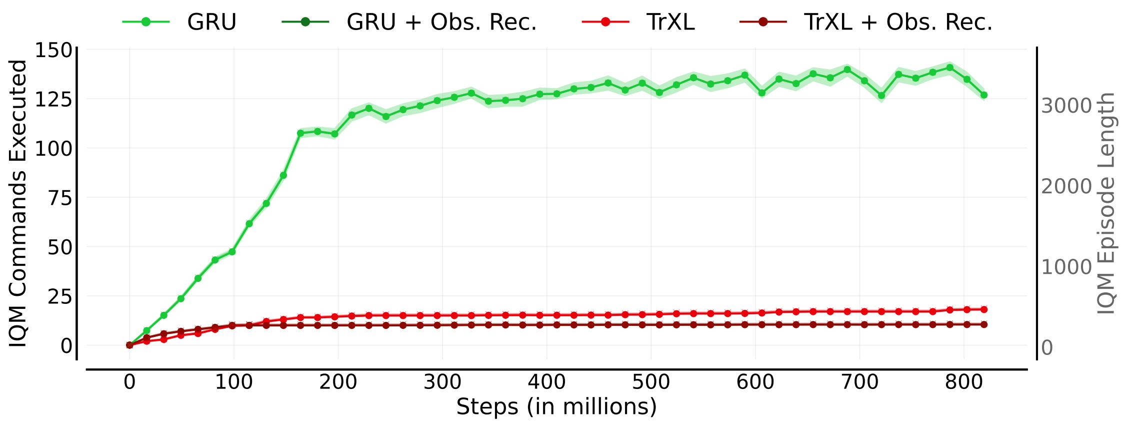
For Mortar Mayhem (Figure 6.2 (a)), the LSTM agent settles at 3 commands, whereas the GTrXL agents using observation reconstruction both achieve an IQM of only 2 commands. In the Mystery Path results (Figure 6.2.3), the LSTM agent achieves full success, albeit requiring more samples compared to the GRU and TrXL agents. The GTrXL agent with a bias of 0 reaches a 94% success rate, while the one with a bias of 2 achieves only 81%, challenging the earlier assertion by Parisotto et al. (2020). The confidence intervals for these results indicate potential training instability. In the Searing Spotlights environment, the LSTM agent with observation reconstruction is slightly less efficient than its GRU counterpart. However, GTrXL agents, though successful, lag behind TrXL with observation reconstruction.

While preliminary, our results suggest that a plug-and-play approach may not always yield effective and sample efficient memory-based agents. However, it is important to note that we avoid making stronger conclusive statements based on these results. This caution stems from the fact that the LSTM and GTrXL baselines are not tuned. In particular, the implementation of GTrXL requires further validation.

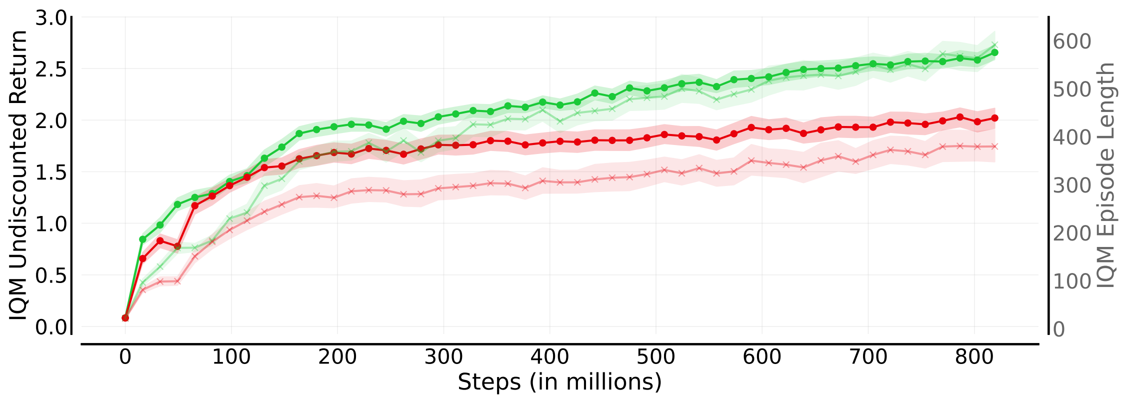
6.3 Endless Environments: GRU Outperforms Transformer-XL

To our surprise, across all three endless environments, the recurrent agent consistently outperforms the transformer-based agent by a significant margin. This diverges from the observations made by Parisotto et al. (2020) and Lampinen et al. (2021), where the LSTM was found to be less effective. Importantly, our TrXL baseline employs absolute positional encoding, deviating from TrXL's original use of relative encoding. Section 7.1 delves deeper into this choice and other potential factors that could account for our contrasting findings.

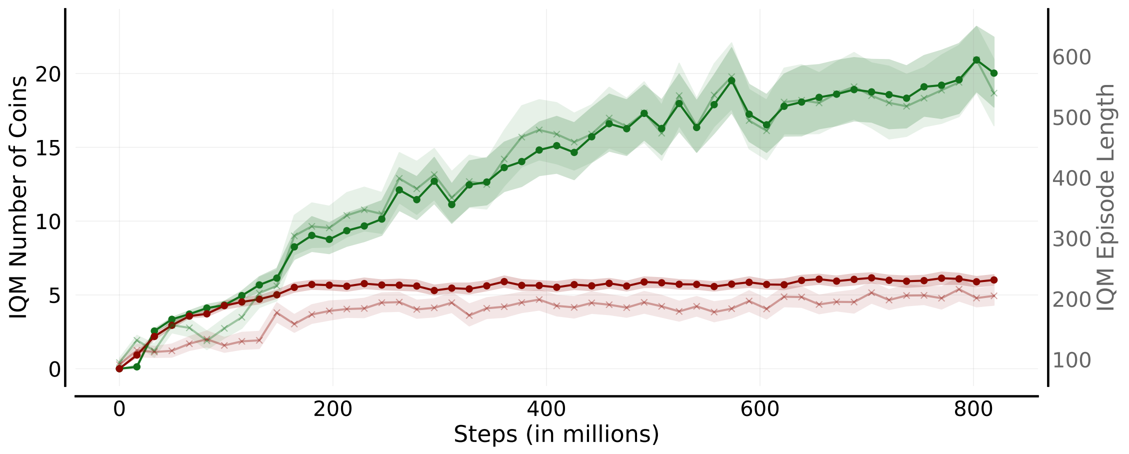
Endless Mortar Mayhem
The most notable gap between GRU and TrXL emerges in the results of Endless Mortar Mayhem (Figure 6.3 (a)). In this context, GRU attains an impressive IQM of 120 executed commands using roughly 300 million steps, whereas TrXL only attains an IQM of 18 commands. Furthermore, the incorporation of observation reconstruction exacerbates this performance difference, leading TrXL to a further deterioration, resulting in an IQM of 10. Furthermore, GRU exhibits consistently longer IQM episode lengths, with TrXL achieving an IQM of 288 steps and GRU reaching a peak IQM episode length of 3462 steps.



(a) Endless Mortar Mayhem



(b) Endless Mystery Path



(c) Endless Searing Spotlights

Figure 6.3: GRU's consistent superiority over TrXL is revealed in Memory Gym's endless environments. The less opaque line marked with crosses depicts the IQM episode length that refers to the right y-axis.

Endless Mystery Path
The outcomes depicted in Figure 6.3 (b) illustrate the results obtained from the Endless Mystery Path environment. In this case, GRU remains more effective than TrXL, although the gap between their performances is narrower compared to Endless Mortar Mayhem. GRU achieves an IQM undiscounted return of 2.65, with TrXL achieving 2.02. Although neither memory approach converges within the allocated training budget of 819 million steps, GRU displays a more pronounced upward trend in performance. In terms of episode length, neither agent achieves lengths as seen in Endless Mortar Mayhem. GRU attains an IQM episode length of 600 steps in Endless Mystery Path, while TrXL reaches up to 400 steps. This divergence can be attributed to the episode's dynamics, where agents tend to fall off after significant progress, necessitating more steps to regain lost ground and accumulate further rewards.

Endless Searing Spotlights
In Endless Searing Spotlights (Figure 6.3 (c)), GRU is about 3.3 times more effective than TrXL. Both agents utilize the observation reconstruction loss, but TrXL converges to an IQM of 6 collected coins after approximately 200 million steps, while GRU achieves an IQM of about 20 coins, continuing to show an upward trend in its performance. The IQM episode length for TrXL is 200 steps, while GRU achieves a longer IQM episode length of 600 steps.

6.4 Wall-time Expenses of Baselines

Relating time dependent measures to sample efficiency comes with the advantage of a consistent scale of time. However, understanding the associated costs of a chosen approach beforehand is crucial. Such details are frequently overlooked in DRL literature. For instance, in our previous work on Memory Gym's finite environments (Pleines et al., 2023), we employed History comprEssion via Language Models (HELM) (Paischer et al., 2022a) as an additional baseline. Given that HELM is at least five times more time-intensive than GRU in terms of wall-time, hyperparameter tuning becomes prohibitively expensive. This overhead originates from the 18 pre-trained and frozen TrXL blocks present in the HELM architecture. Unfortunately, this constraint was not communicated by the original authors, even though we acknowledge HELM's open-source nature and straightforward implementation. Consequently, we shifted our compute budget from HELM to the TrXL baseline, which we consider more insightful and valuable.

Further, a comprehensive evaluation of a new baseline should also consider wall-time. Nevertheless, providing consistent wall-time measurements is often complex due to factors such as dynamic RL environments, hardware variations, software inconsistencies, or varying platform loads. To shed light on the time requirements of the GRU and TrXL baselines, we present data from Endless Mortar Mayhem and the finite version of Mystery Path.

Endless Mortar Mayhem
Table 6.3 presents wall-time efficiency metrics for agents trained on Endless Mortar May-

Table 6.3: Wall-time efficiency metrics for agents trained on Endless Mortar Mayhem. Agents are arranged from left to right based on increasing mean training duration, measured in hours. Other values, excluding the standard deviation, are represented in seconds and comprise the time needed for a single PPO iteration. The second row indicates if the ANN was compiled before training. To this date, PyTorch faces issues in compiling recurrent cells. Note: "TrXL*" metrics use 4,000 initial training data points, while other runs utilize 150k data points covering the full training process. "TrXL + QPos" and "TrXL + LR + QPos + GT" refer to experiments from Section 7.1.

	GRU	TrXL	TrXL + QPos	TrXL + Obs. Rec.	TrXL + LR + QPos + GT	TrXL*
Compiled	No	Yes	Yes	Yes	Yes	No
Mean	4.11	5.53	5.6	5.76	5.86	7.97
Std	0.35	1.12	1.31	1.1	1.68	0.25
Min	3.6	5.2	5	5.4	5.3	7.6
Median	4.1	5.4	5.5	5.7	5.8	7.9
IQM	4.03	5.48	5.53	5.72	5.82	7.93
Mean Duration	57.08	76.81	77.78	80	81.39	110.69

hem, utilizing the noctua2 high-performance cluster[2]. We leverage 32 CPU cores (AMD Milan 7763), an NVIDIA A100 GPU, and 100GB RAM for one training run. With PyTorch version 2, models can be lazily compiled at training onset, reducing TrXL's training time by a significant 30%. Previously, plain TrXL training averaged 110 hours, but this has been reduced to 76 hours. Incorporating observation reconstruction adds roughly 3 hours, and with ground truth estimation, it extends to 81 hours. The GRU agents are the most wall-time efficient, completing in approximately 57 hours. Thus, recurrence emerges as the most effective and efficient architecture in both sample and wall-time metrics for the endless environments.

Mystery Path

We further take a look at the wall-time of agents trained on Mystery Path (Figure 6.4). It becomes apparent that TrXL and GRU are quite on par concerning wall-time efficiency, whereas TrXL needs fewer samples as shown in Section 6.2.2. On first sight, it seems surprising that GTrXL is faster than TrXL (Table 6.4) even though GTrXL is more complex. This is due to the more expensive resets of the environment. Better policies have shorter episodes and thus reset more frequently impairing wall-time. So as GTrXL's and LSTM's policies are inferior, their wall-time is faster. These results were obtained from the LiDo3 high-performance cluster[3]. Each run utilized 32 cores (AMD Epyc 7542), an A100 GPU, and 200GB RAM.

[2]`https://pc2.uni-paderborn.de/de/hpc-services/available-systems/noctua2` Accessed: 2023-09-18

[3]`https://lido.itmc.tu-dortmund.de/` Accessed: 2023-09-18

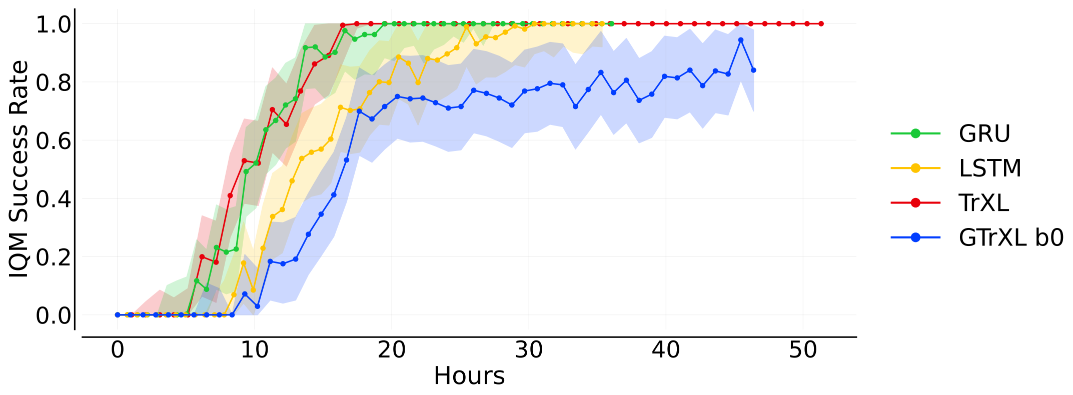


Figure 6.4: Wall-time efficiency curves resulted from experiments in Mystery Path

Table 6.4: Wall-time efficiency metrics for agents trained on Mystery Path. Agents are arranged from left to right based on increasing mean training duration, measured in hours. Other values, excluding the standard deviation, are represented in seconds and comprise the time needed for a single PPO iteration.

	LSTM	GRU	GTrXL B0	TrXL
Compiled	No	No	Yes	Yes
Mean	5.09	5.19	6.68	7.39
Std	0.22	0.28	0.31	0.24
Min	4.2	4.0	5.8	6.4
Max	7.6	7.0	62.2	11.5
Median	5.1	5.2	6.6	7.4
IQM	5.09	5.18	6.64	7.39
Mean Duration	35.35	36.04	46.4	51.32

6.5 Resulted Agent Behaviors

This thesis is complemented by a dedicated website[4], which offers a multi-modal media experience, showcasing various GRU and TrXL agents interacting within Memory Gym's environments. The website provides interactive modalities for visualizing an agent's gameplay, as follows:

- **Videos**:
  - Ground Truth
  - Agent Observation
  - Reconstructed Observation (if applicable)
- **Charts and Plots**:
  - Action Probability Distribution (Bar Chart)
  - State-Value (Line Plot)
  - Entropy (Line Plot)
- **Attention Insights** (exclusively for TrXL):
  - Top or All Attention Scores

For enhanced user experience, data points in the line plots and attention scores are interactive; clicking them navigates the video to the associated time step.

Given the vast array of agents, including their evolving iterations throughout training, it would be impractical to inspecting each in detail. Consequently, we showcase the best-performing final agent from the 5 repetitions. Although the media aids in understanding agents' behaviors, the underlying rationale for specific actions can be opaque due to the inherent black-box nature of ANNs. For transformer models, attention scores provide insights into significant past events. In Mortar Mayhem, distinct duties emerge for each TrXL block. Similar impressions are salient in Mystery Path. Finally, we visualize an agent adeptly dodging spotlights to seize a coin in Endless Searing Spotlights.

6.5.1 Mortar Mayhem: Transformer Blocks Focus Differently

Environments in the Mortar Mayhem family do not leave much room for diverse behaviors as the structure for time and space is fixed. Upon analyzing the attention scores throughout an episode, it is evident that different transformer blocks prioritize distinct time steps, and this varies between the clue and act tasks. Figure 6.5 displays the attention scores at time step 40, corresponding to the end of the clue task. Interestingly, the first two blocks consistently direct their attention to every fourth time step, representing the initial visual frame of a command, while the first block pronounces the episode's outset as well. The last

[4]Agent Behavior Media Website: `https://marcometer.github.io/`

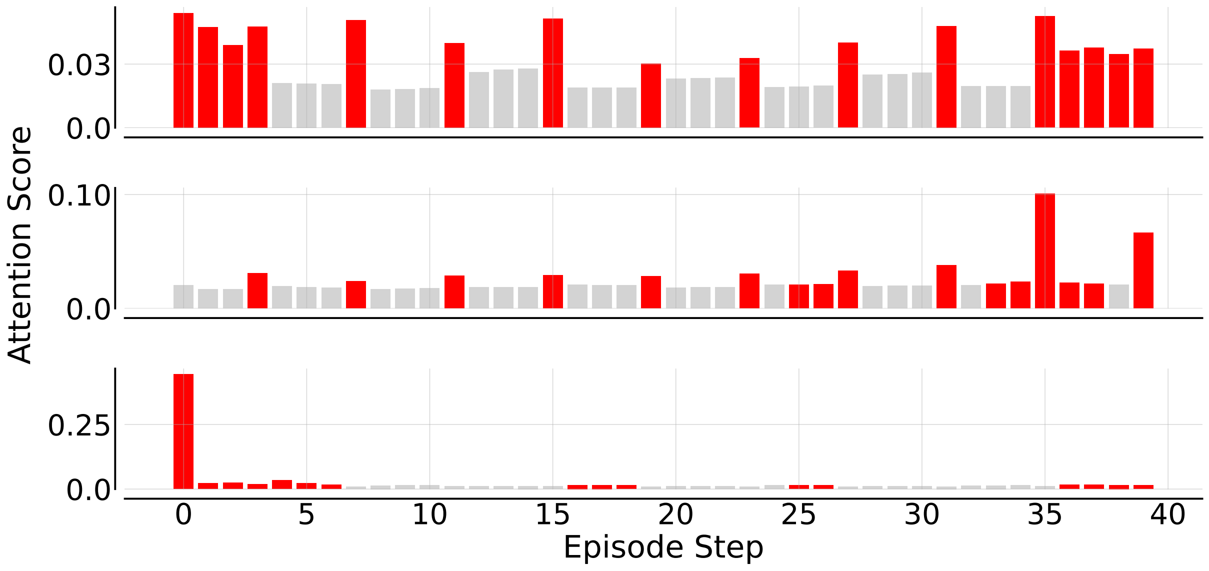


Figure 6.5: Attention scores in Mortar Mayhem at time 40. The top 16 scores are red.

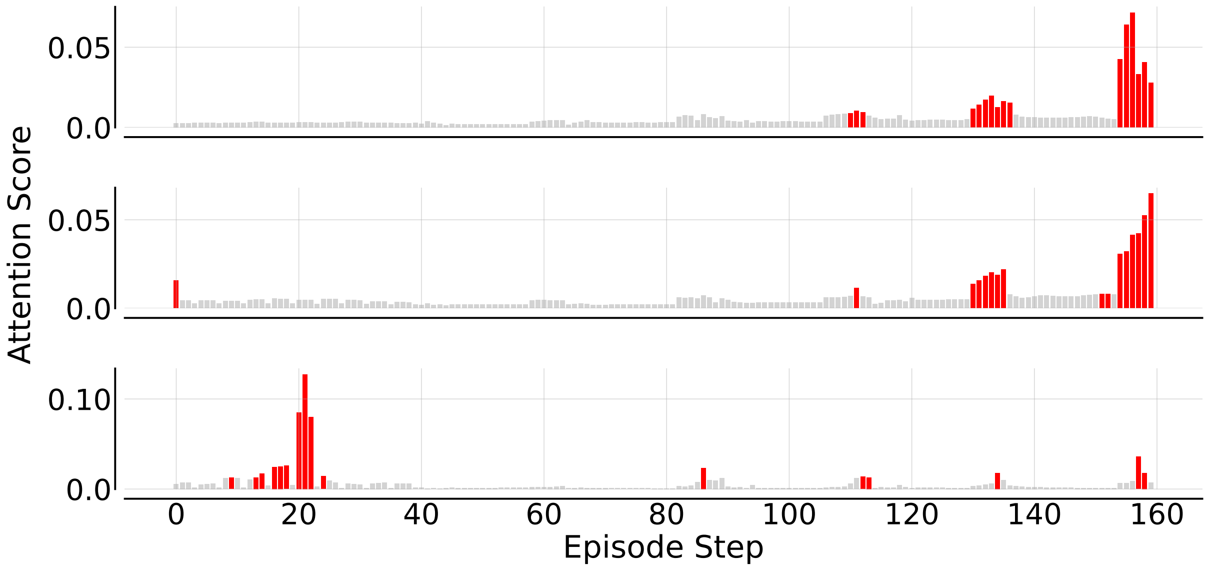


Figure 6.6: Attention scores in Mortar Mayhem at time 160. The top 16 scores are red.

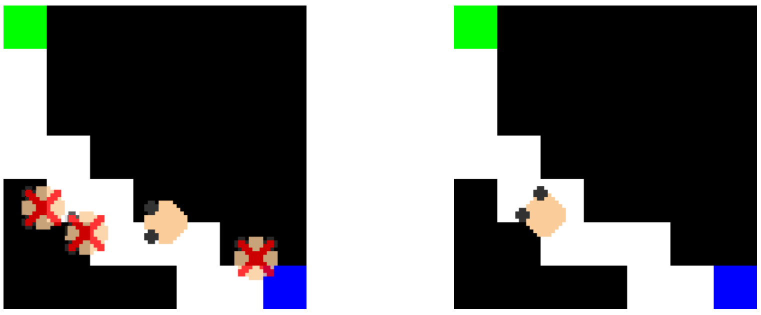

(a) Step 70 (b) Step 50

Figure 6.7: Mystery Path at step 70 (a) and 50 (b). By time 70, the agent has experienced 3 fall offs at steps 5, 28, and 55, indicated by the shaded and crossed agent icons. At step 50 (b), the agent had previously surpassed its progress from step 70 but encountered a fall off just 5 steps afterward.

block emphasizes the episode's initial step. This pattern persists during the clue task but shifts when the agent starts executing commands, as seen in Figure 6.6. For the task of executing the 6th command from time step 160, the third block's primary attention is on time step 21, showcasing the relevant command's visualization. It also marginally focuses on steps 86, 112, 134, and 157, indicative of successful executions, yet it overlooks the first successful command around step 58. Conversely, the other blocks adapt by concentrating more on recent past events.

6.5.2 Mystery Path: Fall Off and Prior Positions Matter

In Mystery Path, even though only the agent is rendered, its starting position upon the episode's beginning serves as a critical cue: the goal likely lies somewhere on the opposite side. In the absence of more specific details, such as fall off locations, in the agent's memory, a logical approach is to first head in that direction. This tactic is commonly adopted by agents across tasks in this category.

Inspecting the attention scores of the TrXL agent at time step 70, as illustrated in Figure 6.7 (a), the first block highlights fall off moments at times 5, 28, and 55, shown in Figure 6.8. Subsequent blocks shift their attention towards events around step 50, recalling a position the agent held 20 steps prior (Figure 6.7 (b)). This earlier knowledge aids the agent in advancing without falling off once more.

Additionally, a performance distinction emerges between the GRU and TrXL agents. In all evaluation episodes and post training, the GRU agent averages 5.65 fall offs, whereas the TrXL agent demonstrates improved efficiency with an average of 4.87.

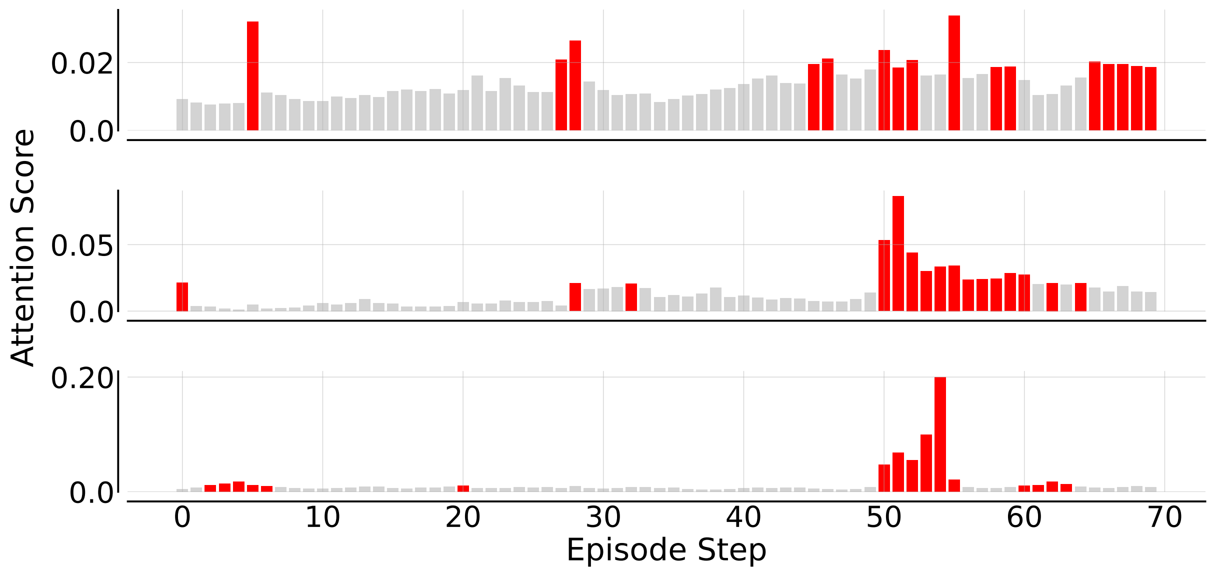


Figure 6.8: Attention scores in Mystery Path at time 70. The top 16 scores are red.

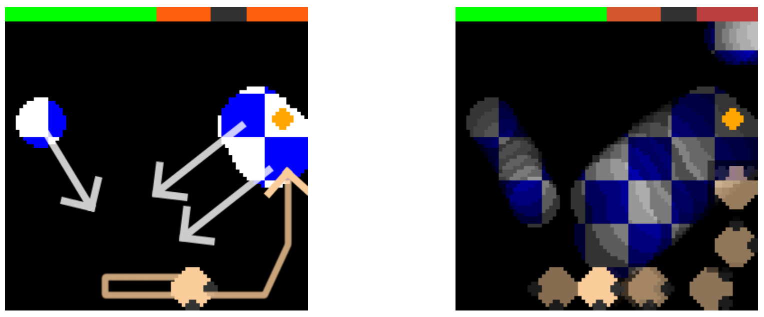

(a) (b)

Figure 6.9: Dodging spotlights in Endless Searing Spotlights. The depicted maneuver, entirely executed in the dark by the TrXL agent, lasts for 64 steps and presents a considerable challenge.

6.5.3 Endless Searing Spotlights: Bypassing Spotlight Traps

In the finite environment, episodes conclude so swiftly that spotlights rarely present a challenge. However, in Endless Searing Spotlights, there are instances where the coin or the agent is ensnared by spotlights. Figure 6.9 illustrates such a scenario. Faced with this, agents have adapted to exhibit patience, biding their time to seize an opening and navigate past the threat. When exploring our media, the GRU[5] and TrXL agents exhibit different maneuvers to solve this instance[6].

6.6 Key Insights: Most Significant Results

To wrap up this chapter, the most salient observation emerges from the endless environments' results: the recurrent agent leveraging GRU significantly surpasses TrXL. Yet, in the finite version of Mortar Mayhem, TrXL solves more commands, and it also poses enhanced efficiency in Mystery Path. During the training in Searing Spotlights, GRU with observation reconstruction requires fewer training samples. Additionally, benchmarking both GTrXL and LSTM without preliminary tuning underscores that simple plug-and-play strategies might not be effective. Finally, insightful agent behaviors were discerned during exploring their behaviors post training; for example, the attention scores of a transformer can provide intuitive understanding of an agent's retrospective reasoning.

[5]GRU: `https://marcometer.github.io/results/ess_gru_rec_seed_200003.html`
[6]TrXL: `https://marcometer.github.io/results/ess_trxl_rec_seed_200003.html`

Chapter 7

Discussion

We presented our GRU and TrXL baselines and unveiled a new set of memory-dependent tasks: Memory Gym. Benchmarking these baselines on Memory Gym in the previous section revealed several insights. Remarkably, the underperformance of TrXL in all endless environments is highly unexpected. Given the prevailing belief that transformers typically surpass recurrent architectures, we discuss five hypotheses. These hypotheses require further in-depth exploration, which is left to future work.

In addition to examining TrXL, we revisit Searing Spotlights. Initially deemed unsolvable by our recurrent baseline, we now discern that the original results were not perturbed by noise due to spotlights. Instead, the normalization of advantage estimates, used to optimize the agent's policy, emerges as the primary factor. Furthermore, we offer a critical assessment of both the finite and endless tasks in Searing Spotlights, pinpointing areas for potential future refinement.

Concluding, we position Memory Gym within the context of a standard benchmark. We advocate that Memory Gym serves as a valuable complement, rather than a comprehensive standard benchmark.

7.1 Transformer-XL's Unexpected Low Effectiveness

The unexpectedly low performance of TrXL prompts us to several hypotheses that are explored within the next subsections:

- The capacity of the agent's model is insufficient.
- The learning signal is too weak.
- The initial query in the first TrXL block lacks temporal information.
- Hidden states within TrXL become stale.
- The absolute positional encoding is indistinguishable.

Endless Mortar Mayhem poses the largest performance gap (Section 6.3), and therefore we utilize this environment for further experiments.

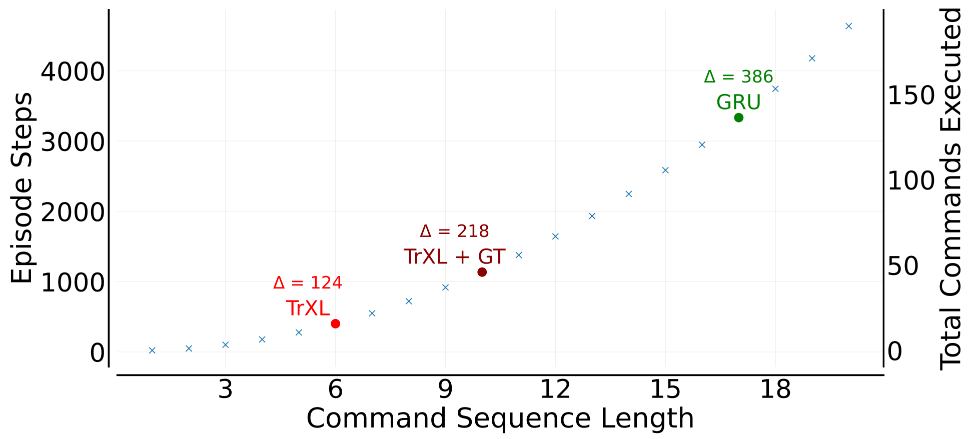


Figure 7.1: Episode steps in Endless Mortar Mayhem increase exponentially with the length of the command sequence to be executed. Data points represent episode time steps where the agent completed the sequence, observed the next command, and subsequently failed. The cumulative successful commands align with the episode's steps and its undiscounted return. The GRU agent's performance is highlighted in green, while the TrXL agent is in red. In addition, we show an agent that illegally includes a ground truth (GT) estimation head as detailed in Section 7.1.2. The delta indicates the step difference to the previous data point, essentially showing how far back the agent has to look into the past. In the default Endless Mortar Mayhem configuration, delta steps grow by 24 for each added command.

7.1.1 Inadequate Model Capacity and Window Length

Our TrXL baseline comprises 2.8 million trainable parameters, while GRU consists of 4.05 million, prompting the question of whether TrXL's model architecture lacks the necessary capacity. To investigate this, we conducted experiments varying the number of blocks (2, 3, 4), and the block's embedding dimension size (256, 384, 512). None of these adjustments led to closing the performance gap to GRU.

We also explored different lengths for the sliding memory window (256, 384, 512). A length of 256 seems theoretically sufficient to rival GRU's performance. This assertion becomes clear when comparing the potential context length of TrXL, enabled by segment-level recurrence, to the lengths needed at certain points in Endless Mortar Mayhem. With a configuration of 256 length and 3 blocks, the recurrent connections can reference information from as far back as $256 + 255 + 255 = 766$ steps. Even though the GRU baseline can handle episodes over 3000 steps, it does not need to recall the episode's beginning. Figure 7.1 illustrates this, detailing both episode lengths and the steps between the complete execution of a command list, which increase incrementally. The TrXL baseline can manage a 5-command sequence, summing to 15 commands. The prior phase of executing 4 commands and observing the new one takes 124 steps. In comparison, the GRU agent has a 386-step gap, which is within the TrXL's context range.

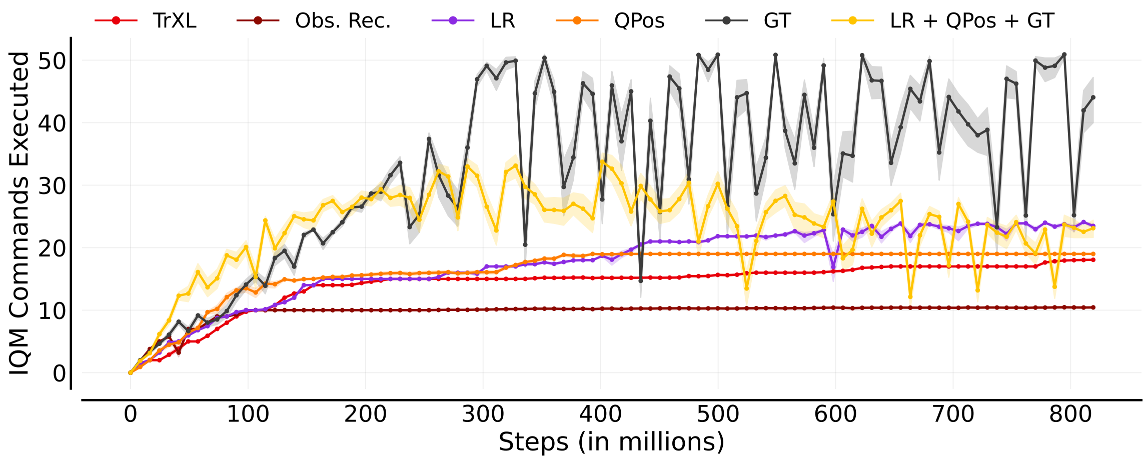


Figure 7.2: Experiments on Endless Mortar Mayhem with varied TrXL configurations. The learning rate schedule (LR) is adjusted to decay from 2.75e-4 to 1.0e-4 over 160 million steps, compared to the previous 1.0e-5. We also incorporate observation reconstruction (Obs. Rec.) and observe the most significant improvement upon introducing a ground truth estimation head (GT) as sanity check. Another test augments TrXL's query with absolute positional encoding (QPos) (Section 7.1.3). The final, but inferior, experiment combines the optimized learning rate, augmented query, and ground truth estimation.

Scaling up the aforementioned dimensions of TrXL has the side effect of an increased demand for GPU memory. Based on our implementation and hyperparameters, this may cause the training to exceed the available 40GB GPU memory of an NVIDIA A100. Workarounds include reducing the batch size or transferring training data between the GPU and CPU, but these options significantly worsen wall-time efficiency.

Furthermore, another indication of untapped capacity lies in the amplification of the learning signal, which we elaborate on next.

7.1.2 Weak Learning Signal

Figure 7.2 depicts several additional experiments aimed at amplifying the learning signal. We made a naive adjustment to the learning rate schedule, utilized observation reconstruction, and included a ground truth estimation head (Section 5.1.3). This ground truth estimation head is added to the actor-critic model architecture and predicts the target position to move to. Labels are provided by the environment, and the estimates are optimized using the mean-squared error loss. While ground truth information is typically unavailable and considered inappropriate for this benchmark, it serves as a useful sanity check. It helps determine if an additional learning signal is advantageous in this context. When training incorporates ground truth estimation, there is a noticeable improvement in the agent's policy. Previously, the IQM of completed commands stood at 18; with ground truth estimation, it reaches 50. This outcome implies that the model's capacity might not be the primary constraint. However, using the ground truth estimation head can lead to instability in training, with the agent's performance fluctuating between 20 and 50 completed commands after about 300 million steps.

During the initial 160 million training steps, the learning rate linearly decays from 2.75e-4 to 1.0e-5. By setting the final decayed learning rate to 1.0e-4, the IQM performance hits 24 commands. Lower performances are observed when adding the observation reconstruction.

In summary, augmenting the learning signal — either by adjusting the learning rate or by incorporating ground truth estimation — leads to more performant policies.

7.1.3 Lack of Temporal Information in the Initial Query

The next hypothesis pertains to the initial query of the first TrXL block. This query is constructed solely based on the features extracted from the observation encoders at the current time step, encompassing only minimal temporal information. In contrast, the query of the second block draws from the aggregated outcome of the memory window, thus capturing more substantial temporal information. However, we believe that the initial query could be further enriched with additional information. In Figure 7.2, one experiment augments the query with absolute positional encoding, providing direct access to the current time step. Despite this enhancement, the agent's performance only moderately improves, reaching an IQM of 19 completed commands. We also explored embedding the original query with the previous output of the last TrXL block, without allowing gradients to back propagate through time. However, this simple approach ended up in catastrophic performances.

All of the aforementioned measures by far do not reach the level of the GRU agent, which peaks at the completion of 140 commands (Figure 6.3 (a)). Even if combined, a low performance of 33 executed commands is achieved, which is also relatively unstable.

7.1.4 Harmful Off-Policy Data in the Episodic Memory

Next, it can be discussed whether the combination of PPO and TrXL's episodic memory pose a threat. PPO is an on-policy algorithm that consumes the training data for a few epochs. Once the second training epoch on the same batch begins, the batch can already be considered off-policy, as the new policy has diverged from the data sampling policy. However, PPO's loss function can mitigate this issue and hence allows for several epochs. But this mitigation is likely limited and we wonder whether this issue is enlarged by the episodic memory.

In our current training setup, an agent collects 512 steps of data, while the TrXL agent, which leverages ground truth estimation, achieves episode lengths greater than 1200 steps. As such episodes exceed the length of the collected data, the earlier hidden states from that episode become outdated by at least one PPO iteration. In future work, it needs to be investigated if stale TrXL hidden states, hinder training progress.

Two approaches are considerable. First of all, the workers could gather trajectories that are twice as long. As this overloads GPU memory, the number of workers need to be halved, which notably increases the wall-time of gathering data. Another approach is to recompute hidden states during each epoch as done in Recurrent Replay Distributed DQN (R2D2) (Kapturowski et al., 2019). Frequently refreshing these come with notable wall-time and GPU memory overheads. The collected training data would need to be re-propagated through the network to refresh the episodic memory. This also requires to maintain all agent observations of an episode across PPO iterations. Efficiently employing TrXL at scale in the context of RL remains as a tremendous challenge.

7.1.5 Indistinguishable Absolute Positional Encoding

The final hypothesis concerns a potential limitation of the absolute positional encoding. Without positional encoding, the agent cannot differentiate the time between past observations. As a default, we rely on absolute positional encodings. This encoding always spans across the potential maximum episode length. In the finite environments, the longest range is set to 512, being the sequence length, which is also used in the original transformer (Vaswani et al., 2017). For the endless tasks, this encoding is set up for 2048 steps, an episode duration that is yet unreachable to the TrXL agents.

The original work of TrXL introduces a relative positional encoding (2.4.6) as a consequence of extended context of lengths. However, when they refer to the absolute positional encoding, they only apply it to the range of the current segment. If a segment has a length of 4, the positions within the segment are always the same (i.e. $[0, 1, 2, 3]$), even though the segments moves across the entire sequence. In our case, the absolute positional encoding considers the entire sequence. If the segment window has progressed in time, the applied positional encoding relies on the current time step (i.e. $[t-3, t-2, t-1, t]$).

Nevertheless, the absolute positional encoding needs to be defined prior training. When scaling it up, the question arises whether the agent's model is capable of differentiating time steps, because a longer length indicates a more fine-grained sample frequency over the underlying positional encoding. To test this, we trained a TrXL agent on the finite task of Mortar Mayhem and configured an absolute positional encoding of length 1024 and 2048. Both experiments, which were repeated twice, resulted in an agent that poorly completes only 1 to 2 commands. This demands for questioning the absolute positional encoding in endless tasks. But if the elements of the absolute position encoding of length 2048 are not distinguishable, how do those agents master more than 20 commands in Endless Mortar Mayhem?

If the assumption holds right that the absolute positional encoding is the culprit in our TrXL baseline, leveraging learned or relative positional encodings instead should improve performance. As seen in Figure 7.3, no notable advancement is accomplished by both variants. We observed two outliers among the 5 runs that utilized relative positional encoding. One underperformed, completing only 13 commands, while the other excelled, finishing more than 60 commands. Since the interquartile mean represents only the central 50% of the data, these outliers are not apparent in the sample efficiency curve.

Given this discussion, a more in-depth exploration of positional encoding in this context is advisable.

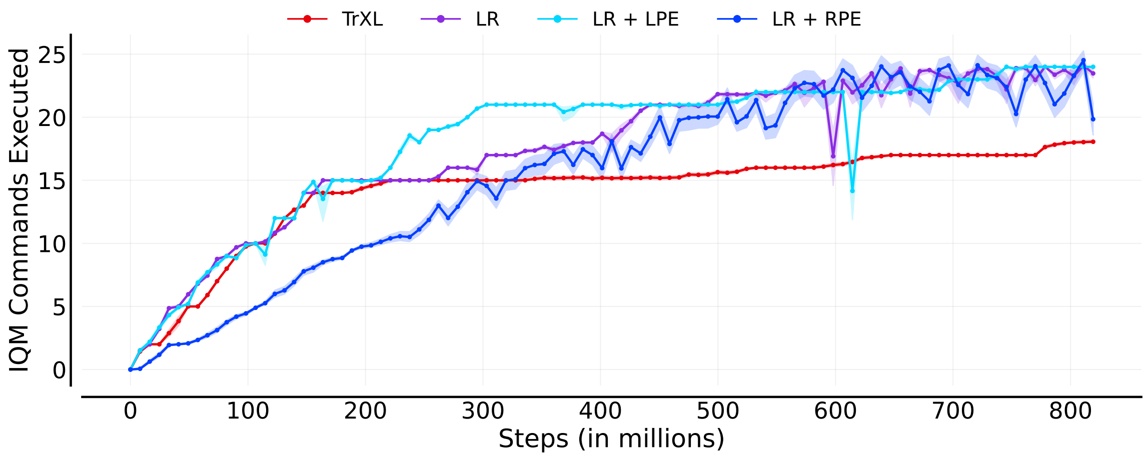


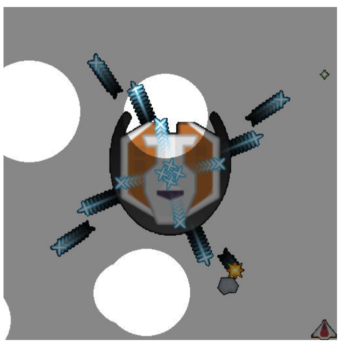
Figure 7.3: Experiments on Endless Mortar Mayhem with varied positional encoding. The learning rate schedule (LR) is adjusted to decay from 2.75e-4 to 1.0e-4 over 160 million steps, compared to the previous 1.0e-5. The acronym LPE refers to a learned positional encoding, while RPE depicts the relative one.

7.2 Recurrence is not Vulnerable to Spotlight Perturbations

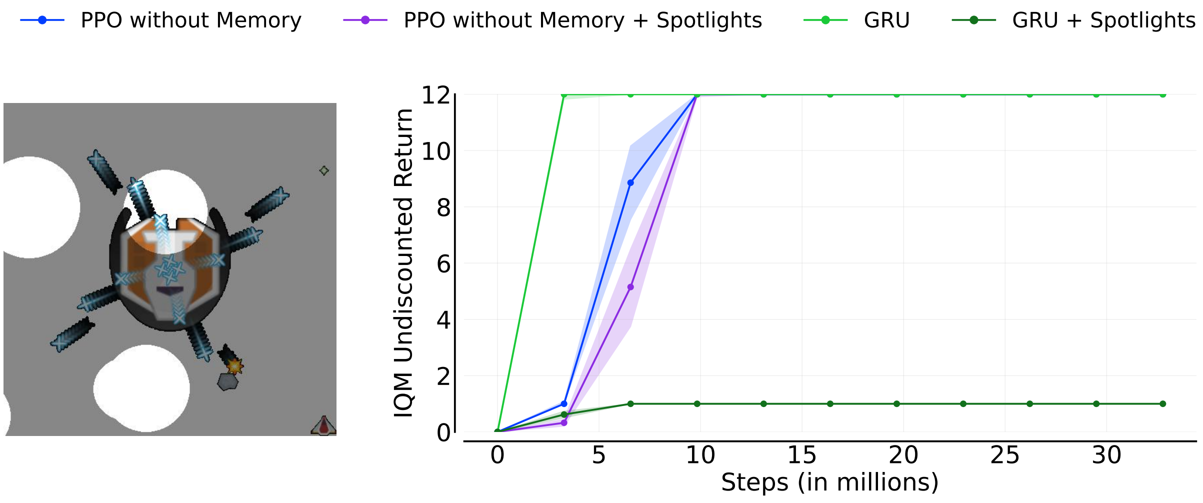
Training recurrent agents in Searing Spotlights with previously established hyperparameters (refer to the last column of Table 6.1) does not yield a meaningful policy. Despite numerous attempts at hyperparameter tuning, architecture adjustments, and task modifications, Searing Spotlights remains as a puzzle. Even with full observability, where all entities are permanently visible, the performance is catastrophic. These challenges prompted the hypothesis that spotlights might introduce noise, heavily affecting the agent's policy (Pleines et al., 2023). Support for this theory come from experiments in the simple BossFight environment (Cobbe et al., 2020), using identical spotlight dynamics as in Searing Spotlights, suggesting a potential vulnerability of recurrent agents to such perturbation.

However, the results presented in Figure 6.1 (g) and 6.2 (c) neglect this hypothesis. This prompted us to conduct an ablation study, beginning with the previous hyperparameters. In this study, only a single hyperparameter or environment option was replaced with the current one at a time. Table 7.1 presents the results for Searing Spotlights when full observability is ensured. Initially, observation reconstruction was presumed to be the solution. However, the crucial adjustment proved to be the decision to avoid normalizing advantage estimates during optimization. Only the GRU agent, which omits advantage normalization, and the memory-less agent, based entirely on previous hyperparameters, recorded noteworthy success rates of 100% and 96%, respectively.

A salient observation emerges when tracking the gradient norms across the agent models during training. With normalized advantages, the gradient norm is typically more than 12 times greater than when normalization is bypassed. This suggests that



(a) Spotlight Perturbation



(b) BossFight Performances on Training Levels

Figure 7.4: GRU and memory-less PPO agents are trained on a single BossFight level (Cobbe et al., 2020), indicating potential sensitivity to spotlight perturbations. Due to the boss's fixed behavior, the environment encourages overfitting, even though the single level is extrapolated to 100,000 using spotlight perturbations (a). Utilizing prior hyperparameters (Table 6.1), the results (b) indicate that while both GRU and memory-less PPO perform well without spotlights, GRU performance drops catastrophically with them present. The IQM undiscounted return represents the total rewards the agent earns in one episode. The agent is depicted by the little space craft in the bottom right corner, while the boss is the mother ship in the center shooting bullets.

Table 7.1: Results from the ablation study on Searing Spotlights with full observability. Each experiment was run 5 times, though the data sourced directly from the training process deviates from our standard protocol. Every PPO iteration (of 5,000 total) produces a data point that reflects the average over the last 100 episodes. Rows are ordered in descending fashion by the mean success rate. The mean is computed from the top-performing data point, averaged across repetitions, from the specific experiment. Consequently, the time point is represented as a percentage duration. The gradient norm is accumulated over all data points within the given duration.

Experiment	**Value**		**Success Rate**		**Duration**	**Gradient Norm**	
	New	Old	Mean	Std		Mean	Std
Advantage Norm.	Off	On	1.00	0.00	31%	0.02	0.01
No Memory			0.96	0.11	99%	0.28	0.03
Agent Health Points	5	10	0.79	0.35	99%	0.30	0.02
Transformer-XL			0.69	0.27	99%	0.30	0.03
Agent Speed	3.0	2.5	0.58	0.22	65%	0.33	0.06
Clip Range	0.1	0.2	0.52	0.25	99%	0.27	0.05
Max Episode Length	256	512	0.48	0.27	99%	0.37	0.04
Old Hyperparameters			0.43	0.06	96%	0.38	0.06
Observation Rec.	On	Off	0.39	0.06	96%	0.40	0.04
Max Gradient Norm	0.25	0.5	0.36	0.05	99%	0.24	0.01

during gradient descent, excessively large optimization steps might be taken, resulting in an oscillating behavior that fails to converge to a local minimum. Simultaneously, unnormalized advantages remain at a scale of 2e-4, but post-normalization, the magnitudes stretch beyond 5. Considering that the policy ratio (as per Equation 2.36) is modulated by these advantage estimates, normalization introduces a strikingly different scale than its absence. Standard practice typically normalizes advantages by subtracting the mean and dividing by the standard deviation, often applied at the mini-batch level.

Several comprehensive studies have delved into such nuances of PPO's implementation (Engstrom et al., 2020; Andrychowicz et al., 2021; Huang et al., 2022a). Among them, only Andrychowicz et al. (2021) turned off advantage normalization and found negligible performance impact. However, their extensive study focused on control tasks with undiscounted returns in the thousands, contrasting Memory Gym's finite environments where the undiscounted rewards does not exceed 2. During hyperparameter optimization, turning off advantage normalization also enhanced performance in Mortar Mayhem and Mystery Path.

Given the critical importance of utilizing raw advantages, there remains an opportunity to delve even deeper into PPO to comprehend and refine its essential aspects.

7.3 Untapped Potential in Searing Spotlights

As demonstrated in Section 3.2.4 and 4.7, Searing Spotlights effectively addresses the desiderata related to memory dependencies. Furthermore, the environment underscores the intricacies in the implementation of PPO, particularly in the context of normalizing advantage estimates. Given these insights, Searing Spotlights and its associated environment families stand as a worthwhile contribution. Nevertheless, a critical reflection reveals opportunities to build upon this robust foundation. In the subsequent sections, we delve into potential refinements, with a focus on the coin collection task in the finite environment, the coin spawn mechanism in the endless environment, and the approach conforming endlessness.

7.3.1 Optimal Agents Swiftly Solve the Finite Task

In Searing Spotlights, optimal agents can complete episodes quickly, averaging 33 steps as shown in Table 4.5. While reaching this optimal behavior is non-trivial due to the challenges posed by spotlights during exploration, the rapid completion diminishes the threat posed by the spotlights (Figure 7.5). To enhance the environment's complexity, agents could be tasked with collecting objects in a specific order. For instance, differently colored coins could indicate the collection sequence. Another idea is to draw inspiration from (Baker et al., 2020), where agents first have to touch each object and then have to revisit them in the same order. Combining this concept with Searing Spotlights' emphasis on darkness offers a potential direction for further refinement.

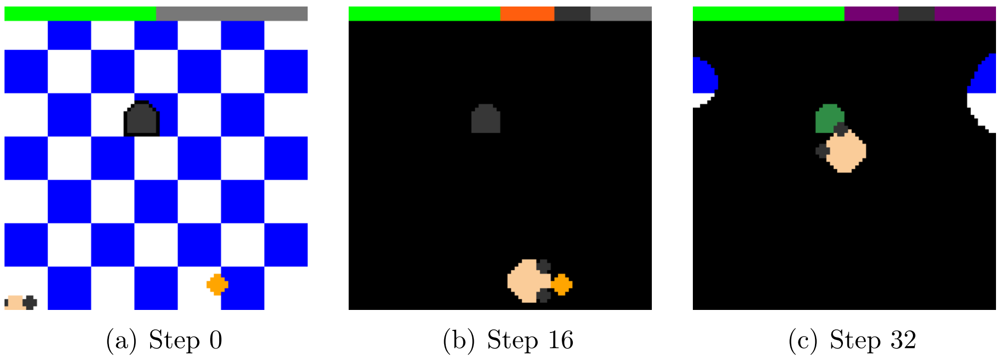


Figure 7.5: In this Searing Spotlights episode, the agent completes the task in 32 steps. By this stage, the spotlights pose no substantial threat to the agent's progress.

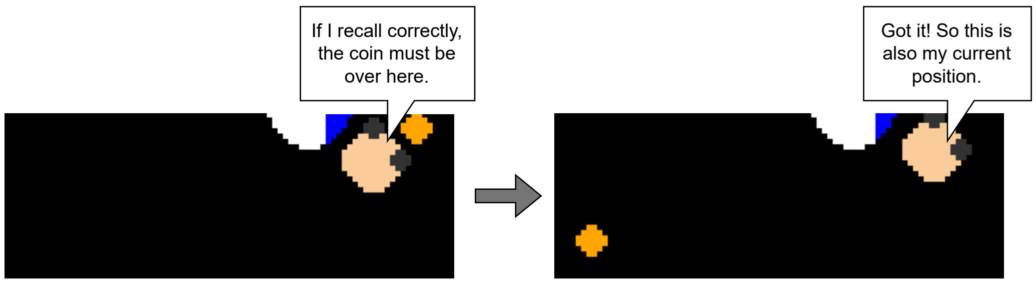

If I recall correctly, the coin must be over here.
Got it! So this is also my current position.

Figure 7.6: In Searing Spotlights, a coin event acts as a positional landmark for the agent. By accurately recalling the coin's position from memory, the agent can refresh its global position and discard outdated information, such as its initial position or past actions.

7.3.2 Refining Coin Spawns in Endless Searing Spotlights

Reflecting on Endless Searing Spotlights, coins spawn anew once collected, and are briefly made visible to the agent. This mechanic ensures that the agent senses the coin's location at least once. However, relying solely on randomly moving spotlights to reveal the coin can be time-consuming. While extending the agent's time is an option, it contradicts the endless nature of the environment where episodes shall terminate as soon as the agent behaves poorly. We propose two ideas for alternative spawning mechanics: spawning the coin within an existing spotlight, and introducing a new spotlight that will inevitably reveal a newly spawned coin. Nonetheless, the first option gives the agent more control over timing the spawn of the new coin, which may be undesirable. The agent could patiently wait for improved spotlight positions to take advantage.

7.3.3 Feedback from Coin Events is a Landmark

In the current setup, agents receive immediate feedback upon collecting a coin, with the new coin being temporarily made visible. This feedback acts as a landmark, as illustrated in Figure 7.6, allowing the agent to orient itself without relying on its initial position. Nonetheless, the endless nature of the task should ideally place a greater memory burden on the agent. To heighten the challenge, the immediate feedback after coin collection should be removed. As previously suggested, the coin could spawn in darkness, with a spotlight eventually revealing it. This mechanism might necessitate added variability in spotlight spawning, such as introducing two spotlights at once or incorporating short, random delays. Alternatively, the coin collection task could be replaced, possibly by rewarding the agent for remaining close to a highlighted spotlight.

7.4 Memory Gym as an Invaluable Complement

The primary objective of this work is to harness the concept of "endlessness" to challenge memory-based DRL agents. While we believe our contribution is both unique and invaluable to the field, we do not aim to establish a definitive standard benchmark. This is a gap that we highlighted in Section 3.3. By "standard benchmark", we mean a set of versatile memory-dependent tasks that has broad acceptance among practitioners. Such widespread acceptance or adaptation of existing memory tasks is not evident among authors who have provided baselines. The reasons become clearer when we analyze these tasks using our proposed desiderata (Section 3.2.2). Although the desiderata primarily address individual tasks rather than collections, the criteria for a standard benchmark remain ambiguous. Defining these criteria for a comprehensive memory task collection necessitates community-wide discussions and in-depth literature reviews spanning the entire memory-based DRL domain. As future advancements of other algorithms might incorporate memory mechanisms, the scope of these reviews may need to extend even further. Without such collaborative and comprehensive efforts, the proposed benchmark risks limited adoption.

It is crucial to underscore that an exhaustive exploration of a comprehensive standard benchmark exceeds this work's scope. Our primary focus is to contextualize Memory Gym, emphasizing that it is rather a unique and valuable complement than a standard benchmark. One key to acceptance is not only the defined task-specific criteria, but also accessibility in a broader sense. Here, accessibility implies the ease of integrating the benchmark with existing baselines and ensuring minimal computational costs. A common facet of this integration involves adapting to the environment's observation and action spaces, topics we will explore subsequently.

7.4.1 On Selecting Widely-Adopted Observation Modalities

Real-world applications present diverse modalities, each demanding unique considerations. For example, an agent might receive observations in the form of vectors, pixels, language, or a mix of these. Consequently, the architecture of an ANN must be tailored to handle potentially multi-modal input streams. However, when evaluating memory capabilities, is it essential for a memory benchmark to accommodate every prevalent modality? While fusing all modalities presents its own set of challenges, it does not inherently test an agent's memory of past observations. Once this fusion hurdle is overcome, what specific challenges do fused modalities pose to an agent's memory?

The need to support various modalities is primarily driven by accessibility. If the majority of DRL implementations are designed for pixel-based observations, a benchmark centered around language observations might reach only a limited community. Adapting to a new modality is not trivial; it can complicate the implementation and potentially alter its behavior. From our experience, either visual or vector-based inputs are commonly utilized, with combinations being less frequent and language-based instructions being rare. Moreover, vector observations are usually less expensive than visual ones.

Memory Gym seamlessly integrates with most setups, primarily focusing on visual observations, which aligns well with the popular Atari CNN encoder (Mnih et al., 2015). However, to enhance Memory Gym's versatility, it could benefit from including task instances based on vector observations. This addition would make Memory Gym's innovative endless tasks more accessible to research labs operating with constrained resources.

7.4.2 The Role of Action Modalities

Observation modalities shape the agent's design, but the selection of an action space is also important. Agents have the flexibility to operate across diverse action spaces, be it discrete, multi-discrete, continuous, language-based, or a hybrid approach. This diversity prompts the question: to what extent do action modalities influence an agent's memory capabilities? Drawing insights from our earlier discussion on observation modalities, we suggest that the influence of action modalities on memory is likely minimal. For example, consider an agent that has been successfully trained on the Mystery Path using multi-discrete actions. The optimized parameters from the visual and memory encoder can be reused for a new agent that operates with continuous actions. During the training of this new agent, these parameters remain unchanged. We anticipate that this new agent will also be successful, given that the previously learned representations from the observation history have proven effective.

In this scenario, it is imperative that the underlying environment supports different action spaces, or the agent has to convert its actions. In the context of Mystery Path's multi-discrete action space, one conversion method is to segment continuous actions into discrete units using a bucketing strategy (Pleines et al., 2019). On the other hand, agents specifically tailored for a single discrete action space can streamline or "flatten" multi-discrete actions, as detailed in Section 5.1.1. It is crucial to understand that such alterations might pose new exploration challenges for the agent. As a result, agents trained from scratch with these adjusted modalities may exhibit performance variations when compared to those in the original multi-discrete action setting.

By converting actions, one can employ baselines rooted in either continuous or discrete actions to train in Memory Gym's environments, which primarily feature a multi-discrete action space. However, the process of discretizing continuous actions may not truly harness the essence of using continuous actions, especially when Memory Gym does not necessitate fine-grained locomotion. When establishing a standard benchmark, it seems crucial to further assess the significance of tasks that naturally rely on continuous actions.

In conclusion, Memory Gym focuses on implementations that accommodate pixel observations and multi-discrete action spaces. Single-dimensional discrete action spaces can be adapted. Although continuous actions can be converted as well, their relevance remains debatable. By offering task variants that solely utilize vector observations, Memory Gym can appeal to a broader audience. As such, we envision Memory Gym at its current state to serve as a valuable complement, distinguished by its endless task nature.

Chapter 8

Conclusion

In this concluding chapter, we revisit the primary challenges faced in memory-based Deep Reinforcement Learning (DRL) and recap our contributions. We then spotlight key insights, placing them within the broader context. Subsequent sections offer recommendations for future research, and we conclude with a critical reflection on our work.

8.1 Recapitulation

Tasks characterized by partial observability or imperfect information in the realm of DRL emphasize the indispensable role of agents with memory capabilities. As we identified, two major challenges loom in this research area:

- Absence of a standard benchmark: Researchers introducing memory-based DRL baselines often employ different tasks to evaluate their agents. This inconsistency might stem from the inaccessibility of certain tasks or the absence of specific qualities that rely on memory-dependence. Evidently, there is a void when it comes to a universally accepted benchmark.
- Limited access to baseline implementations: The broader community often finds itself at a disadvantage as many implementations of DRL algorithms, particularly those incorporating memory mechanisms, are not open source. This opacity is especially noticeable for agents that utilize attention-driven models as transformers. Such withholding impedes reproducibility, subsequently stalling progress in the domain.

Motivated by these challenges, our goal was to foster research in the field by contributing a novel benchmark and two accessible and comprehensible baseline implementations. These are anchored on the widely-adopted Proximal Policy Optimization algorithm. The first baseline harnesses the utility of recurrence through a Gated Recurrent Unit cell, while the second adopts an attention-driven approach, utilizing Transformer-XL as episodic memory with a sliding window approach.

We have introduced a suite of tasks collectively termed "Memory Gym". This suite features the environment families: Mortar Mayhem, Mystery Path, and Searing Spotlights. The finite episode variants of these tasks, which conclude upon achieving a singular objective, align with the criteria set forth in our research. These environments are heavily reliant on memory and necessitate frequent memory interactions. Drawing inspiration from the game "I packed my bag", Memory Gym presents unique endless tasks. Remarkably, these tasks pose an automatic curriculum that progressively challenges an agent's retention and recall capabilities. This allows memory-based agents to showcase their true effectiveness, rather than just their efficiency.

8.2 Key Insights

Because of numerous results (Dai et al., 2019; Parisotto et al., 2020; Hill et al., 2021; Lampinen et al., 2021), a prevailing notion in the field suggests that transformers generally outperform recurrent mechanisms. Our findings from benchmarking the GRU and TrXL agents on Memory Gym's finite tasks largely support this sentiment. Specifically, the TrXL agent demonstrated higher effectiveness in Mortar Mayhem and exhibited greater sample efficiency in Mystery Path. Conversely, the GRU agent displayed enhanced efficiency in Searing Spotlights.

A striking deviation from this trend emerged in the endless environments, where recurrence made a notable comeback. The GRU agent significantly surpassed the performance of the TrXL agent. This unforeseen outcome prompted further exploration of our TrXL baseline. We found TrXL to be more expensive in terms of wall-time and GPU memory overhead, posing limitations to scaling. While it is unlikely that the TrXL suffers from capacity constraints, certain modifications, such as adjusting the learning rate or illegally incorporating ground truth estimation, did yield improvements. Nevertheless, these enhancements were insufficient to match the performance of the GRU agent. Potential explanations for this divergence include the possibility of TrXL missing temporal cues in its initial query, the adverse effects of stale hidden states, and the potential need for more effective positional encoding. These hypotheses present avenues for subsequent research.

In conclusion, Memory Gym presents a unique array of challenges tailored for agents reliant on memory, with its most indistinguishable feature being the endless tasks, behaving as an automatic curriculum. Memory Gym's tasks have played a crucial role in this work, deepening the understanding of memory-based DRL agents. Our TrXL baseline emerges as a noteworthy addition in this context. Though it beckons further investigation, its standout attributes are its accessibility and lucidity. To our knowledge, no other attention-based baseline in the domain of memory-based DRL agents offers such transparency.

8.3 Future Work

In this section, we outline potential directions for future research, emphasizing the refinement of benchmarks and the exploration of prospective memory baselines.

8.3.1 Benchmarking Memory-based Agents

Both the finite and endless versions of the Searing Spotlights tasks have demonstrated their value. The finite version, in particular, shed light on the role of advantage normalization as a culprit for memory-based agents operating under perfect information. However, as elaborated in Section 7.3, there is even further potential for refining these tasks. A longer finite task is commendable since, for an effective agent that completes episodes swiftly, the spotlights no longer pose a threat. Meanwhile, in Endless Searing Spotlights, it might be beneficial to eliminate feedback from coin events. Such feedback aids the agent in reorientation, possibly rendering previous observations dispensable.

In the broader conversation about establishing a universally recognized benchmark, we posit that Memory Gym serves as a valuable complement to the tools for assessing memory capabilities. By transitioning from visual observations to vector ones, Memory Gym can cater to a broader audience and be more accommodating to limited computational resources. In this vein, introducing tasks tailored for continuous action spaces could further enhance Memory Gym's utility. However, it is essential to prioritize the quality of these tasks over quantity, ensuring they are meaningfully crafted, accessible, and accompanied by comprehensive documentation.

Lastly, accelerating the simulation speed of Memory Gym's environments would be a significant advancement. In fields as robotics, environments can be simulated using GPUs, facilitating massive parallelization and consequently faster training (Makoviychuk et al., 2021). While adapting GPUs for environment simulation, especially with rendered visual observations, presents challenges, the potential benefits make it a worthy endeavor.

8.3.2 Expanding and Enhancing Memory Baselines

Our initial findings, derived from benchmarking Memory Gym using the contributed GRU and TrXL baselines, set the stage for further exploration. It would be compelling to assess how other recent advancements in the field, such as HCAM (Lampinen et al., 2021) and HELMv2 (Paischer et al., 2022b), perform on Memory Gym. Such evaluations would enable a more comprehensive comparison among these methods.

There are also opportunities to refine our TrXL baseline. By investigating the hypotheses related to the TrXL's initial query, the potential drawbacks of stale hidden states, and the nuances of positional encoding, the performance of this baseline could be enhanced. Additionally, exploring other transformer architectures might be beneficial, especially when considering the challenges of managing overheads during scaling. An intriguing avenue could be the Expire-Span method (Sukhbaatar et al., 2021), a transformer variant designed to "expire" or forget outdated information, allowing it to handle extended contexts effectively.

Beyond attention-based and recurrent models, the exploration of structured state space sequence models (Gu et al., 2022) might be fruitful, especially in light of their recent debut in DRL (Lu et al., 2023). State space models provide a mathematical framework that captures the dynamics between observed data and hidden states over time. They can be implemented in a convolutional fashion, which benefits from enhanced parallelization, or in a recurrent fashion, optimized for fast inference. As research progresses, it will be fascinating to observe whether the Gated Recurrent Unit maintains its leading position on Memory Gym's unique endless tasks.

8.4 Critical Reflection

Our scope in this work spans two primary domains: benchmarks and baselines. Each domain, while deserving its own in-depth exploration, presents a unique set of challenges that often intersect. Developing a benchmark necessitates the presence of accessible and working baselines to effectively assert criteria and implementation details. Conversely, the introduction of novel baselines demands a robust set of tasks to validate their efficacy. Given the existing gaps in both areas, we found ourselves navigating these intertwined challenges, often facing decisions that required careful balancing and trade-offs.

Upon reflection, there might have been opportunities to streamline the development of our baselines and environments. An early adoption of vector observations might have made the development of the environments less computationally expensive, while this measure would have enriched Memory Gym's versatility as well. Additionally, it is plausible that grayscale visual observations could convey as much information as RGB pixels, potentially reducing both computational demands and training durations. These saved resources could have been redirected to delve deeper into Memory Gym's intricacies. For instance, examining the environments' difficulty levels could have provided richer insights. Similarly, paralleling the findings of Cobbe et al. (2020), we could have evaluated the required number of unique levels for effective generalization.

In retrospection of our contributed baselines, we recognize that while the TrXL heavily relies on relative positional encoding, our experiments predominantly employed absolute positional encodings. Allocating more resources to explore this aspect might have provided clearer insights into the TrXL's potential. Regarding the recurrent agent, which did not receive as much attention as the TrXL one, a direct comparison with related implementations could have been enlightening, potentially uncovering strengths or areas for improvement. This is particularly pertinent given the critical impact of implementation details in DRL.

Despite these critical thoughts, we are confident that Memory Gym and our baselines represent a significant advancement in the realm of memory-based DRL agents. The recognition of our finite tasks and the recurrent baseline, as evidenced by their acceptance at the renowned International Conference on Learning Representations (Pleines et al., 2023), underscores their value. Our introduction of the innovative endless concepts of ever-growing difficulties, coupled with the open-source TrXL baseline, further amplifies our contribution. This extension, in the form of a journal paper, is currently under review.

References

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis, editors, *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, pages 2623–2631. ACM, 2019. doi: 10.1145/3292500.3330701. URL `https://doi.org/10.1145/3292500.3330701`.

Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *Int. J. Robotics Res.*, 39(1), 2020. doi: 10.1177/0278364919887447. URL `https://doi.org/10.1177/0278364919887447`.

Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Leonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What matters for on-policy deep actor-critic methods? a large-scale study. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=nIAxjsniDzg`.

Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016. URL `http://arxiv.org/abs/1607.06450`.

Bowen Baker, Ingmar Kanitscheider, Todor M. Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=SkxpxJBKwS`.

Charles Beattie, Joel Z. Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, Julian Schrittwieser, Keith Anderson, Sarah York, Max Cant, Adam Cain, Adrian Bolton, Stephen Gaffney, Helen King, Demis Hassabis, Shane Legg, and Stig Petersen. Deepmind lab. *CoRR*, abs/1612.03801, 2016. URL `http://arxiv.org/abs/1612.03801`.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.*, 47: 253–279, 2013. doi: 10.1613/jair.3912. URL `https://doi.org/10.1613/jair.3912`.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*, pages 2546–2554, 2011. URL `https://proceedings.neurips.cc/paper/2011/hash/86e8f7ab32cfd12577bc2619bc635690-Abstract.html`.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019. URL `http://arxiv.org/abs/1912.06680`.

Maxime Chevalier-Boisvert. Miniworld: Minimalistic 3d environment for rl & robotics research. `https://github.com/maximecb/gym-miniworld`, 2018. Accessed: 2023-09-18.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018. Accessed: 2023-09-18.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734. ACL, 2014. doi: 10.3115/v1/d14-1179. URL `https://doi.org/10.3115/v1/d14-1179`.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL `http://arxiv.org/abs/1412.3555`.

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 2048–2056. PMLR, 2020. URL `http://proceedings.mlr.press/v119/cobbe20a.html`.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc Viet Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, *Proceedings* *of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 2978–2988. Association for Computational Linguistics, 2019. doi: 10.18653/v1/p19-1285. URL `https://doi.org/10.18653/v1/p19-1285`.

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de las Casas, Craig Donner, Leslie Fritz, Cristian Galperti, Andrea Huber, James Keeling, Maria Tsimpoukelli, Jackie Kay, Antoine Merle, Jean-Marc Moret, Seb Noury, Federico Pesamosca, David Pfau, Olivier Sauter, Cristian Sommariva, Stefano Coda, Basil Duval, Ambrogio Fasoli, Pushmeet Kohli, Koray Kavukcuoglu, Demis Hassabis, and Martin Riedmiller. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602 (7897):414–419, Feb 2022. ISSN 1476-4687. doi: 10.1038/s41586-021-04301-9. URL `https://doi.org/10.1038/s41586-021-04301-9`.

DI-engine Contributors. DI-engine: OpenDILab decision intelligence engine. `https://github.com/opendilab/DI-engine`, 2021. Accessed: 2023-09-18.

Philipp Dufter, Martin Schmitt, and Hinrich Schütze. Position Information in Transformers: An Overview. *Computational Linguistics*, 48(3):733–763, 09 2022. ISSN 0891-2017. doi: 10.1162/coli_a_00445. URL `https://doi.org/10.1162/coli_a_00445`.

B. Efron. Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, 7 (1):1 – 26, 1979. doi: 10.1214/aos/1176344552. URL `https://doi.org/10.1214/aos/1176344552`.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. ISSN 0364-0213. doi: https://doi.org/10.1016/0364-0213(90)90002-E. URL `https://www.sciencedirect.com/science/article/pii/036402139090002E`.

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep RL: A case study on PPO and TRPO. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=r1etN1rtPB`.

Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1406–1415. PMLR, 2018. URL `http://proceedings.mlr.press/v80/espeholt18a.html`.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi

Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. In *NeurIPS*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/74a67268c5cc5910f64938cac4526a90-Abstract-Datasets_and_Benchmarks.html.

Meire Fortunato, Melissa Tan, Ryan Faulkner, Steven Hansen, Adrià Puigdomènech Badia, Gavin Buttimore, Charles Deck, Joel Z Leibo, and Charles Blundell. Generalization of reinforcement learners with working and episodic memory. In *Advances in Neural Information Processing Systems*, pages 12448–12457, 2019.

Kunihiko Fukushima. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, 5(4): 322–333, 1969. doi: 10.1109/TSSC.1969.300225.

Felix A. Gers, Nicol N. Schraudolph, and Jürgen Schmidhuber. Learning precise timing with LSTM recurrent networks. *J. Mach. Learn. Res.*, 3:115–143, 2002. URL http://jmlr.org/papers/v3/gers02a.html.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and D. Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, volume 9 of *JMLR Proceedings*, pages 249–256. JMLR.org, 2010. URL http://proceedings.mlr.press/v9/glorot10a.html.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

L. Graesser and W.L. Keng. *Foundations of Deep Reinforcement Learning: Theory and Practice in Python*. Addison-Wesley data and analytics series. Addison-Wesley, 2020. ISBN 9780135172384. URL https://books.google.de/books?id=QVA8ugEACAAJ.

Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*, volume 385 of *Studies in Computational Intelligence*. Springer, 2012. ISBN 978-3-642-24796-5. doi: 10.1007/978-3-642-24797-2. URL https://doi.org/10.1007/978-3-642-24797-2.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=uYLFoz1vlAC.

Çaglar Gülçehre, Tom Le Paine, Bobak Shahriari, Misha Denil, Matt Hoffman, Hubert Soyer, Richard Tanburn, Steven Kapturowski, Neil C. Rabinowitz, Duncan Williams, Gabriel Barth-Maron, Ziyu Wang, Nando de Freitas, and Worlds Team. Making efficient use of demonstrations to solve hard exploration problems. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020a. URL https://openreview.net/forum?id=SygKyeHKDH.

Çaglar Gülçehre, Tom Le Paine, Bobak Shahriari, Misha Denil, Matt Hoffman, Hubert Soyer, Richard Tanburn, Steven Kapturowski, Neil C. Rabinowitz, Duncan Williams, Gabriel Barth-Maron, Ziyu Wang, Nando de Freitas, and Worlds Team. Making efficient use of demonstrations to solve hard exploration problems. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020b. URL `https://openreview.net/forum?id=SygKyeHKDH`.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1856–1865. PMLR, 2018. URL `http://proceedings.mlr.press/v80/haarnoja18b.html`.

Danijar Hafner. Benchmarking the spectrum of agent capabilities. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=1W0z96MFEoH`.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.*, 4(2):100–107, 1968. doi: 10.1109/TSSC.1968.300136. URL `https://doi.org/10.1109/TSSC.1968.300136`.

Matthew J. Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposia, Arlington, Virginia, USA, November 12-14, 2015*, pages 29–37. AAAI Press, 2015. URL `http://www.aaai.org/ocs/index.php/FSS/FSS15/paper/view/11673`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1026–1034. IEEE Computer Society, 2015. doi: 10.1109/ICCV.2015.123. URL `https://doi.org/10.1109/ICCV.2015.123`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.90. URL `https://doi.org/10.1109/CVPR.2016.90`.

Nicolas Heess, Jonathan J. Hunt, Timothy P. Lillicrap, and David Silver. Memory-based control with recurrent neural networks. *CoRR*, abs/1512.04455, 2015. URL `http://arxiv.org/abs/1512.04455`.

Felix Hill, Olivier Tieleman, Tamara von Glehn, Nathaniel Wong, Hamza Merzic, and Stephen Clark. Grounded language learning fast and slow. In *9th International Confer-*

ence on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net, 2021. URL https://openreview.net/forum?id=wpSWuz_hyqA.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

Shengyi Huang, Rousslan Fernand Julien Dossa, Antonin Raffin, Anssi Kanervisto, and Weixun Wang. The 37 implementation details of proximal policy optimization. In *ICLR Blog Track*, 2022a. URL https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/. Accessed: 2023-09-18.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23 (274):1–18, 2022b. URL http://jmlr.org/papers/v23/21-1342.html.

Xiao Shi Huang, Felipe Pérez, Jimmy Ba, and Maksims Volkovs. Improving transformer optimization through better initialization. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 4475–4483. PMLR, 2020. URL http://proceedings.mlr.press/v119/huang20f.html.

Jan Humplik, Alexandre Galashov, Leonard Hasenclever, Pedro A. Ortega, Yee Whye Teh, and Nicolas Heess. Meta reinforcement learning as task inference. *CoRR*, abs/1905.06424, 2019. URL http://arxiv.org/abs/1905.06424.

Max Jaderberg, Wojciech M. Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castañeda, Charles Beattie, Neil C. Rabinowitz, Ari S. Morcos, Avraham Ruderman, Nicolas Sonnerat, Tim Green, Louise Deason, Joel Z. Leibo, David Silver, Demis Hassabis, Koray Kavukcuoglu, and Thore Graepel. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443): 859–865, 2019. doi: 10.1126/science.aau6249. URL https://www.science.org/doi/abs/10.1126/science.aau6249.

G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer Texts in Statistics. Springer US, 2021. ISBN 9781071614181. URL https://books.google.de/books?id=5dQ6EAAAQBAJ.

Michael I. Jordan. Chapter 25 - serial order: A parallel distributed processing approach. In John W. Donahoe and Vivian Packard Dorsel, editors, *Neural-Network Models of Cognition*, volume 121 of *Advances in Psychology*, pages 471–495. North-

Holland, 1997. doi: https://doi.org/10.1016/S0166-4115(97)80111-2. URL https://www.sciencedirect.com/science/article/pii/S0166411597801112.

Arthur Juliani, Ahmed Khalifa, Vincent-Pierre Berges, Jonathan Harper, Ervin Teng, Hunter Henry, Adam Crespi, Julian Togelius, and Danny Lange. Obstacle tower: A generalization challenge in vision, control, and planning. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 2684–2691. ijcai.org, 2019. doi: 10.24963/ijcai.2019/373. URL https://doi.org/10.24963/ijcai.2019/373.

Sham M. Kakade and John Langford. Approximately optimal approximate reinforcement learning. In Claude Sammut and Achim G. Hoffmann, editors, *Machine Learning, Proceedings of the Nineteenth International Conference (ICML 2002), University of New South Wales, Sydney, Australia, July 8-12, 2002*, pages 267–274. Morgan Kaufmann, 2002.

Steven Kapturowski, Georg Ostrovski, Will Dabney, John Quan, and Remi Munos. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=r1lyTjAqYX.

Gaspard Lambrechts, Adrien Bolland, and Damien Ernst. Recurrent networks, hidden states and beliefs in partially observable environments. *Trans. Mach. Learn. Res.*, 2022, 2022. URL https://openreview.net/forum?id=dkHfV3wB2l.

Andrew Kyle Lampinen, Stephanie C.Y. Chan, Andrea Banino, and Felix Hill. Towards mental time travel: a hierarchical memory for reinforcement learning agents. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=wfiVgITyCC_.

Donghoon Lee, Taehwan Kwon, Seungeun Rho, Daniel Wontae Nam, Jongmin Kim, Daejin Jo, and Sungwoong Kim. Brain agent. https://github.com/kakaobrain/brain_agent, 2022. Accessed: 2023-09-18.

Joel Z. Leibo, Cyprien de Masson d'Autume, Daniel Zoran, David Amos, Charles Beattie, Keith Anderson, Antonio García Castañeda, Manuel Sanchez, Simon Green, Audrunas Gruslys, Shane Legg, Demis Hassabis, and Matthew M. Botvinick. Psychlab: A psychology laboratory for deep reinforcement learning agents. *CoRR*, abs/1801.08116, 2018. URL http://arxiv.org/abs/1801.08116.

Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. RLlib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2018.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Chris Lu, Yannick Schroecker, Albert Gu, Emilio Parisotto, Jakob Nicolaus Foerster, Satinder Singh, and Feryal Behbahani. Structured state space models for in-context reinforcement learning. In *ICML Workshop on New Frontiers in Learning, Control, and Dynamical Systems*, 2023. URL https://openreview.net/forum?id=CKPTz21e6k.

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance GPU based physics simulation for robot learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=fgFBtYgJQX_.

Luckeciano C. Melo. Transformers are meta-reinforcement learners. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 15340–15359. PMLR, 2022. URL https://proceedings.mlr.press/v162/melo22a.html.

Lingheng Meng, Rob Gorbet, and Dana Kulic. Memory-based deep reinforcement learning for pomdps. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2021, Prague, Czech Republic, September 27 - Oct. 1, 2021*, pages 5619–5626. IEEE, 2021. doi: 10.1109/IROS51168.2021.9636140. URL https://doi.org/10.1109/IROS51168.2021.9636140.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Byj72udxe.

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe W. J. Jiang, Ebrahim M. Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Anand Babu, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. Chip placement with deep reinforcement learning. *CoRR*, abs/2004.10746, 2020. URL https://arxiv.org/abs/2004.10746.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nat.*, 518(7540):529–533, 2015. doi: 10.1038/nature14236. URL https://doi.org/10.1038/nature14236.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1928–1937. JMLR.org, 2016. URL http://proceedings.mlr.press/v48/mniha16.html.

Steven Morad, Ryan Kortvelesy, Matteo Bettini, Stephan Liwicki, and Amanda Prorok. POPGym: Benchmarking partially observable reinforcement learning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=chDrutUTs0K.

M. Morales. *Grokking Deep Reinforcement Learning*. Manning Publications, 2020. ISBN 9781617295454. URL https://books.google.de/books?id=IpHJzAEACAAJ.

Richard G.M. Morris. Spatial localization does not require the presence of local cues. *Learning and Motivation*, 12(2):239–260, 1981. ISSN 0023-9690. doi: https://doi.org/10.1016/0023-9690(81)90020-5. URL https://www.sciencedirect.com/science/article/pii/0023969081900205.

Tianwei Ni, Benjamin Eysenbach, and Ruslan Salakhutdinov. Recurrent model-free RL can be a strong baseline for many pomdps. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 16691–16723. PMLR, 2022. URL https://proceedings.mlr.press/v162/ni22a.html.

OpenAI. Introducing chatgpt. https://openai.com/blog/chatgpt, 2022. Accessed: 2023-09-18.

Fabian Paischer, Thomas Adler, Vihang P. Patil, Angela Bitto-Nemling, Markus Holzleitner, Sebastian Lehner, Hamid Eghbal-Zadeh, and Sepp Hochreiter. History compression via language models in reinforcement learning. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 17156–17185. PMLR, 2022a. URL https://proceedings.mlr.press/v162/paischer22a.html.

Fabian Paischer, Thomas Adler, Andreas Radler, Markus Hofmarcher, and Sepp Hochreiter. Toward semantic history compression for reinforcement learning. In *Second Workshop on Language and Reinforcement Learning*, 2022b. URL https://openreview.net/forum?id=97C6klf5shp.

Emilio Parisotto and Ruslan Salakhutdinov. Efficient transformers in reinforcement learning using actor-learner distillation. In *9th International Conference on Learning*

Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net, 2021. URL https://openreview.net/forum?id=uR9LaO_QxF.

Emilio Parisotto, H. Francis Song, Jack W. Rae, Razvan Pascanu, Çaglar Gülçehre, Siddhant M. Jayakumar, Max Jaderberg, Raphaël Lopez Kaufman, Aidan Clark, Seb Noury, Matthew M. Botvinick, Nicolas Heess, and Raia Hadsell. Stabilizing transformers for reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 7487–7498. PMLR, 2020. URL http://proceedings.mlr.press/v119/parisotto20a.html.

Marco Pleines, Frank Zimmer, and Vincent-Pierre Berges. Action spaces in deep reinforcement learning to mimic human input devices. In *2019 IEEE Conference on Games (CoG)*, pages 1–8, 2019. doi: 10.1109/CIG.2019.8848080.

Marco Pleines, Matthias Pallasch, Frank Zimmer, and Mike Preuss. Generalization, mayhems and limits in recurrent proximal policy optimization. *CoRR*, abs/2205.11104, 2022. doi: 10.48550/arXiv.2205.11104. URL https://doi.org/10.48550/arXiv.2205.11104.

Marco Pleines, Matthias Pallasch, Frank Zimmer, and Mike Preuss. Memory gym: Partially observable challenges to memory-based agents. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=jHc8dCx6DDr.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. ISSN 1476-4687. doi: 10.1038/323533a0. URL https://doi.org/10.1038/323533a0.

John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. Trust region policy optimization. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 1889–1897. JMLR.org, 2015.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1506.02438.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Noor Shaker, Julian Togelius, and Mark J. Nelson. *Procedural Content Generation in Games*. Computational Synthesis and Creative Systems. Springer, 2016. ISBN 978-

3-319-42714-0. doi: 10.1007/978-3-319-42716-4. URL https://doi.org/10.1007/978-3-319-42716-4.

Wenling Shang, Xiaofei Wang, Aravind Srinivas, Aravind Rajeswaran, Yang Gao, Pieter Abbeel, and Michael Laskin. Reinforcement learning with latent flow. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 22171–22183, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/ba3c5fe1d6d6708b5bffaeb6942b7e04-Abstract.html.

C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x.

H. Francis Song, Abbas Abdolmaleki, Jost Tobias Springenberg, Aidan Clark, Hubert Soyer, Jack W. Rae, Seb Noury, Arun Ahuja, Siqi Liu, Dhruva Tirumala, Nicolas Heess, Dan Belov, Martin A. Riedmiller, and Matthew M. Botvinick. V-MPO: on-policy maximum a posteriori policy optimization for discrete and continuous control. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=SylOlp4FvH.

Sainbayar Sukhbaatar, Da Ju, Spencer Poff, Stephen Roller, Arthur Szlam, Jason Weston, and Angela Fan. Not all memories are created equal: Learning to forget by expiring. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 9902–9912. PMLR, 2021. URL http://proceedings.mlr.press/v139/sukhbaatar21a.html.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018. URL http://incompleteideas.net/book/the-book.html.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena : A benchmark for efficient transformers. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=qVyeW-grC2k.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Comput. Surv.*, 55(6):109:1–109:28, 2023. doi: 10.1145/3530811. URL https://doi.org/10.1145/3530811.

Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023. URL https://zenodo.org/record/8127025.

Andrew Trask. *Grokking Deep Learning.* Manning Publications Co., USA, 1st edition, 2019. ISBN 1617293709.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017. URL `https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html`.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom Le Paine, Çaglar Gülçehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft II using multi-agent reinforcement learning. *Nat.*, 575(7782):350–354, 2019. doi: 10.1038/s41586-019-1724-z. URL `https://doi.org/10.1038/s41586-019-1724-z`.

Jane Wang, Michael King, Nicolas Porcel, Zeb Kurth-Nelson, Tina Zhu, Charlie Deck, Peter Choy, Mary Cassin, Malcolm Reynolds, Francis Song, Gavin Buttimore, David Reichert, Neil Rabinowitz, Loic Matthey, Demis Hassabis, Alex Lerchner, and Matthew Botvinick. Alchemy: A structured task distribution for meta-reinforcement learning. *arXiv preprint arXiv:2102.02926*, 2021. URL `https://arxiv.org/abs/2102.02926`.

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Daan Wierstra, Alexander Förster, Jan Peters, and Jürgen Schmidhuber. Solving deep memory pomdps with recurrent policy gradients. In Joaquim Marques de Sá, Luís A. Alexandre, Wlodzislaw Duch, and Danilo P. Mandic, editors, *Artificial Neural Networks - ICANN 2007, 17th International Conference, Porto, Portugal, September 9-13, 2007, Proceedings, Part I*, volume 4668 of *Lecture Notes in Computer Science*, pages 697–706. Springer, 2007. doi: 10.1007/978-3-540-74690-4_71. URL `https://doi.org/10.1007/978-3-540-74690-4_71`.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

Marek Wydmuch, Michaλ Kempka, and Wojciech Jaśkowski. Vizdoom competitions: Playing doom from pixels. *IEEE Transactions on Games*, 2018.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 10524–10533. PMLR, 2020. URL `http://proceedings.mlr.press/v119/xiong20b.html`.

Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A review of recurrent neural networks: LSTM cells and network architectures. *Neural Comput.*, 31(7):1235–1270, 2019. doi: 10.1162/neco_a_01199. URL `https://doi.org/10.1162/neco_a_01199`.

List of Acronyms

A2C Advantage Actor-Critic. 4, 15, 16, 20, 21, 23, 24

A3C Asynchronus Advantage Actor-Critic. 23, 42, 43

ANN Artificial Neural Network. 3, 7, 26, 81, 82, 84, 93, 95, 111

APE Additive Positional Encoding. 36, 40

BPTT Back-Propagation Through Time. 28, 29

CI Confidence Interval. 85

CNN Convolutional Neural Network. 26, 51, 70, 80, 83, 112

CPI Conservative Policy Iteration. 21

DL Deep Learning. 2, 63

DM DeepMind. 44, 47–51, 57, 58, 134

DMLab-30 DeepMind Lab. 42, 43, 48, 51, 63

DQN Deep Q-Network. 7, 23, 41, 42

DRL Deep Reinforcement Learning. 1–4, 7, 8, 15, 26–28, 32, 40–42, 44, 48, 50, 52, 69, 71, 72, 76, 78, 81, 85, 92, 111, 112, 115–118, 134

DRQN Deep Recurrent Q-Network. 43

FPS Frames per Second. 48

GAE Generalized Advantage Estimation. 17, 20, 21, 24

GRU Gated Recurrent Unit. 1, 3, 4, 7, 26, 30–32, 40, 43, 44, 52, 63, 69, 70, 73, 74, 76, 77, 80, 81, 83–86, 88, 90–93, 95, 97, 99, 101, 102, 104, 106, 107, 116, 117, 134

GTrXL Gated Transformer-XL. 4, 7, 32, 40, 52, 69, 77, 80, 81, 85, 86, 89, 90, 93, 99, 134

HCAM Hierarchical Chunk Attention Mechanism. 69, 77, 117

HELM History comprEssion via Language Models. 92, 117

HPC High-Performance Computing. 46

IQM Interquartile Mean. 85, 88, 90–92, 103, 104, 107

LSTM Long Short-Term Memory. 4, 7, 26, 29–32, 42, 43, 52, 69, 73, 77, 81, 86, 89, 90, 93, 99, 134

MAM Modifying Attention Matrix. 40

MC Monte Carlo. 12, 13, 18, 24

MDP Markov Decision Process. 7–9

MHA Multi-Head Attention. 32, 34, 37, 40, 79

MLP Multi-Layer Perceptron. 26, 29, 36, 70

MSE Mean Squared Error. 19, 72

POMDP Partially Observable Markov Decision Process. 7, 10, 11, 44

PPO Proximal Policy Optimization. 3, 4, 7, 15–24, 52, 69, 73, 78, 84–86, 93, 94, 104, 105, 107–109

R2D2 Recurrent Replay Distributed DQN. 105

ReLU Rectified Linear Unit. 77, 83

RL Reinforcement Learning. 7, 8, 10, 14, 15, 17, 22, 40, 45, 64, 75, 77, 92, 105

RNN Recurrent Neural Network. 3, 26–29, 32, 41, 50–52, 69, 73, 74, 134

TD Temporal Difference. 13, 19, 20

TrXL Transformer-XL. 3–5, 7, 32, 37–40, 43, 63, 69, 70, 77–81, 83–86, 88, 90–93, 95, 97–99, 101–105, 116–118, 134, 135

List of Figures

List of Tables

List of Algorithms