# Preferential Proximal Policy Optimization in Reinforcement Learning

by

Tamilselvan Balasuntharam

A thesis submitted to the
School of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the degree of

**Master of Science**

**in**

**Computer Science**

Faculty of Science
University of Ontario Institute of Technology
(Ontario Tech University)

Oshawa, Ontario, Canada
December 2023

## THESIS EXAMINATION INFORMATION

Submitted by: Tamilselvan Balasuntharam

Master of Science in Computer Science

Thesis title: Preferential Proximal Policy Optimization in Reinforcement Learning

An oral defense of this thesis took place on November 24, 2023 in front of the following examining committee:

| | |
|---|---|
| Chair of Examining Committee | Dr. Loutfouz Zaman |
| Research Supervisor | Dr. Kourosh Davoudi |
| Co-Research Supervisor | Dr. Mehran Ebrahimi |
| Examining Committee Member | Dr. Ali Neshati |
| Thesis Examiner | Dr. Akramul Azim |

The above committee determined that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies.

# Abstract

The Proximal Policy Optimization (PPO), a policy gradient method, excels in reinforcement learning with its "surrogate" objective function and stochastic gradient ascent. However, PPO does not fully consider the significance of frequently encountered states in policy/value updates. To address this, this Thesis introduces Preferential Proximal Policy Optimization (P3O), which integrates the importance of these states into parameter updates. We determine state importance by multiplying the variance of action probabilities by the value function, then normalizing and smoothing this with the Exponentially Weighted Moving Average (EWMA). This calculated importance is incorporated into the surrogate objective function, redefining value and advantage estimation in PPO. Our method auto-selects state importance, which can apply to any on-policy reinforcement learning algorithm using a value function. Empirical evaluations across six Atari environments demonstrate that our approach outperforms the baseline (vanilla PPO) across different tested environments, highlighting the value of our proposed method in learning complex environments.

**Keywords:** Reinforcement Learning; Policy Gradient Methods; Deep Learning; Policy Optimization

# Author's Declaration

I hereby declare that this submission is entirely my own work, in my own words, and that all sources used in researching it are fully acknowledged. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners. I authorize the University of Ontario Institute of Technology (Ontario Tech University) to lend this thesis to other institutions or individuals for the purpose of scholarly research. I further authorize University of Ontario Institute of Technology (Ontario Tech University) to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research. I understand that my thesis will be made electronically available to the public.

_____ Tamilselvan Balasuntharam

# Statement of Contribution

I hereby certify that I am the sole author of this thesis and that part of this thesis has been accepted at International Conference on Machine Learning and Applications (ICMLA). I have used standard referencing practices to acknowledge ideas, research techniques, or other materials that belong to others. Furthermore, I hereby certify that I am the sole source of the creative works and/or inventive knowledge described in this thesis.

# Acknowledgements

I express my profound gratitude to my family, friends, and advisors for their unwavering support. Their encouragement has been instrumental in my pursuit of advanced studies and has shaped my consideration for a future PhD. Their influence over the years has been invaluable, and I am deeply appreciative of the positive impact they've had on my personal and academic growth. Thank you.

# Contents

# Acronyms

**A2C** Advantage Actor Critic.

**AI** Artificial Intelligence.

**DQN** Deep Q-Network.

**EWMA** Exponentially Weighted Moving Average.

**GAE** Generalized Advantage Estimate.

**GBAE** Generalized Beta Advantage Estimate.

**GPU** Graphics Processing Unit.

**MDP** Markov Decision Process.

**ML** Machine Learning.

**P3O** Preferential Proximal Policy Optimization.

**PPG** Phasic Policy Gradient.

**PPO** Proximal Policy Optimization.

**PTD** Preferential Temporal Difference.

**RL** Reinforcement Learning.

**SI** State Importance.

**TD-learning** Temporal Difference learning.

**VPG** Vanilla Policy Gradient.

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Introduction

Over recent decades, advancements in computational resources have had an extraordinary impact in Machine Learning (ML). In particular the augmented capabilities of Graphics Processing Unit (GPU) have fortified deep learning, making it as an essential technological in recent times. At the forefront of this Artificial Intelligence (AI) revolution there is a sub-domain of ML called Reinforcement Learning (RL).

RL has had major advances in recent years [34]. The field can interact with many complex games in the Atari 2600 system [28] and defeat professional players in games like Dota 2 [6] and StarCraft [44]. RL has grown to a point where it can excel in many tasks such as robotics [19], healthcare [52], news recommendation [26], stock trading [51], and many more using traditional algorithms such as advantage actor-critic (A2C) [27], Vanilla Policy Gradient (VPG) [38] and Proximal Policy Optimization (PPO) [33].

RL distinguishes itself within the machine learning standard by its independence from labeled training data [36]. Instead of relying on predefined labels, RL algorithms

learn through interactions experienced within a specific environment. The entity that engages with and learns from this environment is termed an 'agent', which could be a computational system or even a robot. The environment, on the other hand, can span a spectrum from virtual simulations and games to tangible real-world settings. To illustrate, consider an autonomous vehicle scenario: the agent is the vehicle's onboard computer system, while the environment encompasses the real-world roads, traffic, and conditions. At discrete time intervals, the agent must decide on actions, such as acceleration, steering, or braking, based on its current state and understanding of the environment.

In RL, the primary objective of an agent is to optimize its total reward within a given environment. Consequently, the design of the reward function becomes important, as it directly influences the agent's behavior and learning trajectory. Crafting this function is often entrusted to human experts, making it one of the most intricate aspects of achieving optimal RL algorithm performance. Given the profound impact of the reward function on agent efficacy, its formulation typically requires deep domain-specific knowledge and expertise.

RL can be classified into two main categories, *model-based* and *model-free* RL [36]. Model-based RL requires a model to capture the transition between one state to another, while model-free methods operate without explicit knowledge of these transition probabilities. For scenarios with vast and intricate state spaces, such as the previously mentioned car example, model-based approaches may become infeasible due to the sheer complexity of modeling every known state. Model-free algorithms instead engage directly from the environment, collecting experiences to determine what is the transition between one state to another. However, a trade-off exists: these algorithms require extensive experiences to grasp knowledge from their interactions,

often requiring repeating experiences to similar situations [36].

Model-free algorithms can be further categorized into two distinct groups: The first, *off-policy* [36] methods, which learn the value of the optimal policy irrespective of the agent's current policy. Unlike off-policy methods, which can learn about an optimal policy while behaving according to a different policy, *on-policy* methods both learn from and act according to the same policy, refining it over time. Many on-policy algorithms, especially actor-critic methods, utilize both a policy (the one selecting the actions), denoted as $\pi(s)$, and a value function, represented as $V(s)$, for any given state $s$ [36].

The value function $V(s)$ is used to estimate the expected reward that the agent will retrieve during training, which in turn, guide agents in selecting appropriate actions. However, not all states possess equal significance. Recently, Anand et al. [1] proposed Preferential Temporal Difference (PTD) Learning that employs a state-dependent preference (i.e., importance) function, denoted as $\beta(s)$, to update the value function according to each state's perceived importance. Consequently, states with low preference experience infrequent updates and reduced likelihood of being employed for updating the target, while those with high preference witness more frequent updates and exert a greater influence on the update. This model can also be applicable in real life. For example if using the car example mentioned earlier, a state where you are driving on a street where no cars and pedestrians are present does not have the same importance compared to a busy intersection with multiple cars and people are on the street. In most RL algorithms, these states are treated the same but in reality they are different. Anand et al. [1] demonstrated that this state-dependent preference technique can enhance the effectiveness of the value function in RL by selectively emphasizing high-value states and suppressing low-value states.

The idea of certain states are more important than others is notably important for complex environments such as the Atari 2600 games [4] where there are numerous different states and actions to consider compared to a simple environment like the cart-pole environment [8]. By finding and updating the value function based on these high-value states, PTD potentially could enable more efficient and effective learning in environments, which could ultimately lead to better performance on tasks like playing Atari games.

Previous methodologies calculated beta values by accessing simple states that only had few features, as demonstrated in Table 4.2 for the Cart Pole environment. Calculating beta states requires knowledge of the environment and internal observations, such as cart pole position, velocity, and angle, as described by Anand et al. [1]. This is practical for environments where internal observations are available as well as having a small amount of observation space. However, in situations where environments provide solely images (e.g., the Atari environment [8]) or offer a multitude of diverse observations (e.g., the MuJoCo environments [41]), identifying beta states becomes infeasible.

To that end, we introduce a novel Preferential Proximal Policy optimization (P3O) algorithm equipping vanilla PPO with a mechanism leveraging state importance. First, inspired by [22], we determine state importance ($SI$) for state $s$. Then, we smooth $SI$ values using Exponentially Weighted Moving Average [18] and normalize the values, generating $\beta(s)$. This allows $\beta(s)$ to be any number between 0 and 1, as opposed to fixed beta values, such as 0.1 or 1 used in the PTD approach [1].

Subsequently, we modify the Generalized Advantage Estimation (GAE) algorithm [32] to incorporate $\beta(s)$, yielding the Generalized Beta Advantage Estimation (GBAE). This modification results in higher bias and lower variance for the advantage estimate

if the current state is high-value and vice versa for low-value states which in turn lead to potential increase in returns as shown in the Experiment Section 5. Bias in RL is the consistent error in predictions, while variance is the inconsistency due to the model's sensitivity to different experiences. This Bias-Variance trade-off is important as it is an important piece of achieving good generalization in RL [36].

Lastly, we integrate the automatically computed $\beta(s)$ values and GBAE into the Proximal Policy Optimization (PPO) algorithm [33], creating the Preferential Proximal Policy Optimization (P3O) algorithm. In particular, P3O replaces traditional advantages with $\beta$-advantages and updates the value function according to $\beta(s)$ values. We compare P3O with PPO to demonstrate that our approach achieves better performance in the Atari environments [4].

Our contributions are summarized as follows:

1. We incorporate state importance ($\beta(s)$) into the GAE algorithm to create the GBAE algorithm.

2. We propose a Preferential Proximal Policy Optimization (P3O) approach introducing state importance and advantage calculated based on GBAE.

3. We develop an automated process for determining beta values, thus eliminating the need for manual configuration to determine beta states based on a threshold.

4. We demonstrate the efficacy of proposed P3O in image-based environments, highlighting the versatility of the algorithm across diverse settings.

Our algorithm is not limited to PPO and could be applied to other on-policy Reinforcement Learning (RL) algorithms. The proposed algorithm provides a framework towards preferential learning for multiple different algorithms as well. This means

that our framework could be placed towards algorithms such as Advantage Actor Critic (A2C) [27], or newer on-policy algorithms like Phasic Policy Gradient (PPG) [10].

# Chapter 2

# Background

## 2.1 Agent-Environment Formulation

RL distinguishes itself from other paradigms in machine learning as it neither depends on labeled data like supervised learning nor on unlabeled data as in unsupervised learning. Instead, RL aims to establish mappings from situations to actions with the overarching goal of maximizing a cumulative numerical reward signal. Crucially, the RL agent doesn't receive explicit directives about which actions to pursue. Instead, it autonomously explores the action space to identify those that result in the maximum reward over time. This intrinsic exploration strategy incorporates two fundamental RL characteristics: *trial-and-error* search and the influence of *delayed* rewards. These characteristics highlight that chosen actions can affect not just immediate rewards but also the potential sequence of rewards in subsequent time-steps.

To methodically capture the nuances of RL, we employ the framework of the Markov Decision Process (MDP) characterized by a discrete action space $A$ and state space $S$. MDPs offer a formalization of sequential decision-making, encompassing the influence of actions on both immediate and future rewards. Essentially, MDPs serve

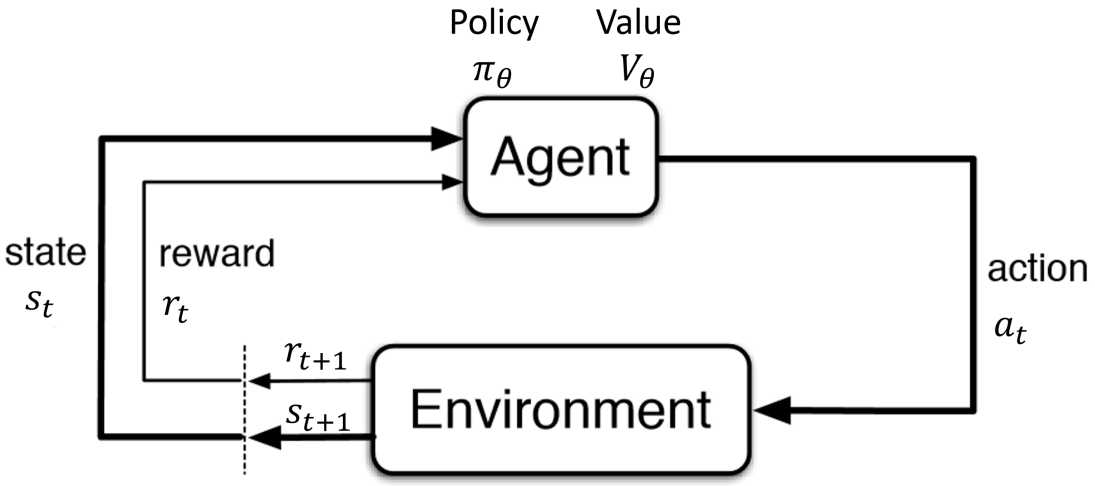



Figure 2.1: This figure represents how the agent interacts with the environment. The agent will send an action $A_t$, to the environment and the environment will send a state of where the agent has ended up when taking the action $S_t$ and a reward signal $R_t$. The reward will help the agent to improve its performance based on the feedback since the agent's goal is to maximize the amount of reward and update the on-policy agent's parameters $\theta$ for the Value and Policy functions $V$ and $\pi$ respectively.

as an elegant mathematical representation of the RL challenges, setting a foundation for RL [36].

In this formulation, the decision Markov entity is referred to as the "agent". The agent operates within an "environment" which is everything outside the agent's control [36]. Interaction within the environment involves the agent selecting actions based on its internal policy, which, in turn, influences the environment. Following each action, the environment not only transitions to a new state but also provides feedback in the form of a reward corresponding to the action's efficacy. The choice of action space is often derived from the nature of the environment.

When the agent is in a state $s_t \in S$, it selects an action $a_t \in A$ based on its prevailing policy, denoted as $\pi$. Post-action, the environment yields a reward $r_{t+1}$, dependent on the current state $s_t$ and the executed action $a_t$, where $t$ signifies

the current time-step. The agent's objective lies in iteratively refining the network parameters $\theta$ for the Value $V$ and Policy $\pi$ functions, trying to obtain the optimal $\theta$ configuration that maximizes the expected return. The return $G_t$ is derived from a function of the reward sequence. A common example of this is the discounted return, given by:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \tag{2.1}$$

In this equation, $r_t$ represents the reward at the current time-step, $r_{t+1}$ is the reward at the next time-step, and $\gamma$ is the discount factor, typically set around 0.99.

A comprehensive visual representation of this agent-environment interaction is depicted in Figure 2.1 [36].

## 2.2 The Exploration-Exploitation Trade-off in Reinforcement Learning

A fundamental challenge in RL is managing the balance between *exploration* and *exploitation*. Exploration is a phase where the agent explores multiple different actions to determine which actions are better than others. Exploitation is the phase where the agent will take the current best action it has learned from experience. This means that the ideal scenario is to have the agent explore and experienced multiple different actions for a particular state and then exploit the optimal action at that state. Successfully navigating this trade-off is essential for an agent to learn an effective policy. The crux of this challenge lies in the dual needs of the agent: to experiment with different actions to understand their potential rewards and to leverage its current knowledge to

secure immediate rewards.

The choice between exploring unfamiliar actions or exploiting known actions is often termed the exploration-exploitation dilemma [36]. The approach an agent adopts with respect to this dilemma can significantly shape its learning process and final performance. Over the years, various algorithms have been introduced to address this pivotal challenge.

For instance, the Deep Q-Network (DQN) introduced by [28] employs an *epsilon-greedy* exploration strategy. This method uses a parameter, $\epsilon$, to dictate the balance between exploration and exploitation. Initially, with $\epsilon$ set near 1, the agent predominantly explores the environment. As the agent gathers knowledge and experience, $\epsilon$ diminishes, pushing the agent towards a more exploit-oriented strategy.

Another prominent method is the Proximal Policy Optimization (PPO). In PPO, an entropy is used in the decision-making process. A high entropy suggests greater uncertainty in the action choices, thus favoring exploration. In contrast, lower entropy drives the agent to exploit known strategies. By integrating this entropy term $s(\pi)$ into the update function, PPO offers a refined approach to exploration, helping avoid early convergence to less optimal policies.

## 2.3 Policies and Value Functions

State-of-the-art RL methodologies predominantly hinge on the precise estimation of value functions. These functions, delineated across states or state-action pairs, equip the agent with insights into the anticipated returns upon executing action $a$ within state $s$ at a designated time-step $t$. Two primary types of value functions are prevalent in RL: the State-Value function $V^{\pi}(s)$ and the Action-Value function $Q^{\pi}(s, a)$ where

$\pi$ denotes the action chosen from the policy. These functions are defined policies that guide the agent's action-selection process. Policies can be either deterministic or stochastic.

Formally, a policy $\pi$ maps states to action probabilities. When an agent, at time $t$, adheres to a policy $\pi$, $\pi(a_t|s_t)$ represents the conditional probability that $a_t$ given $s_t$ [36]. Essentially, a policy dictates the agent's strategy for selecting the next action based on the current state. A deterministic policy assigns a specific action to each state, whereas a stochastic policy provides a probability distribution over potential actions from which an action is sampled. For instance, in a simple grid-world scenario, a deterministic policy might direct the agent to consistently move right from a certain state. In contrast, a stochastic policy in the same environment could propose an 80% likelihood of moving right and a 15% chance of moving up or a 5% chance of doing nothing. In our algorithm we used the State-Value function $V^{\pi}(s)$ which is defined as [36],

$$V^{\pi}(s) = E_{\pi}\{R_t|s_t = s\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s\} \tag{2.2}$$

## 2.4 Policy Gradient Algorithms

Policy gradient methods are approaches to solve RL problems. In particular, since the goal of RL algorithms is to find optimal behaviour strategy that will help the agent to achieve the highest reward, the policy gradient methods model and optimize the policy directly. Policy gradient algorithms, including Advantage Actor Critic (A2C) [27], PPO [33], and Phasic Policy Gradient (PPG) [10], update the policy based on the current policy distribution. All experiences used to update the policy are derived

from the current policy unlike off-policy methods where they update from previous experiences under different policies. The policy gradient can be expressed as:

$$\nabla_\theta J(\theta) = E[\sum_{t=0}^{\infty} \nabla_\theta log \pi_\theta(a_t|s_t) G_t]. \tag{2.3}$$

Where $\nabla_\theta$ represents the gradient of the object function $J(\theta)$ with respect to the parameter $\theta$, where $a_t$ is the action $a$ at time-step $t$ and $s_t$ is state at time-step $t$.

This expression is not limited to $G_t$ and can use the TD advantage $\delta_t = r_t + V^\pi(s_{t+1}) - V^\pi(s_t)$, Advantage function $A^\pi(s_t, a_t)$ (Eq 2.4), Generalized Advantage Estimation (Generalized Advantage Estimate (GAE)) $\hat{A}_t^{GAE(\gamma,\lambda)}$ (Discussed in 4.3), or the return $G_t^\beta$ (2.8) in Preferential Temporal Difference (PTD) Learning approach to either reduce and improve variance within the algorithm.

The goal in policy gradient methods is to maximize the expected total reward by repeatedly estimating the gradient.

## 2.5 Generalized Advantage Estimation

Policy gradient algorithms, in their raw form, offer unbiased estimates but suffer from high variance. To address this, advantage functions, denoted by $A(a, s)$, have been employed to assess how an action deviates in quality from the average action for a given state. However, directly estimating the Advantage function poses challenges. It is essentially the difference between the Q-function $Q(s, a)$ and the value function $V(s)$, both of which are prone to high variance. Formally, this relationship is given as

$$A(s, a) = Q(s, a) - V(s). \tag{2.4}$$

Such variance in estimates can destabilize training since both the $Q(s, a)$ and $V(s)$ are estimated from data. The estimates directly affect the computation of the advantage function and if $V(S)$ and $Q(s, a)$ are not estimated correctly, it can cause training stability issues in learning. To mitigate these challenges, Schulman et al. [1] introduced the Generalized Advantage Estimation (GAE) [32]. The GAE formulation is given by

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V = \delta_t + (\gamma\lambda) A_{t+1}^{GAE}, \tag{2.5}$$

where $\gamma$ typically assumes a value close to 0.99, representing the discount factor. $\lambda$ serves as a steeper discount factor, constrained within the interval $0 \leq \lambda \leq 1$. The term $\delta$ signifies the advantage, described as $r_t + \gamma V(s_{t+1}) - V(s_t)$ at time-step $t + l$ for the value function $V$. $t$ represents the current time step and $l$ is the offset that ranges from 0 to infinity.

While GAE introduces a degree of bias, it significantly curtails variance through the introduction of a decay parameter $\lambda$. This decay parameter offers a balance between bias and variance: when $\lambda = 0$, the estimator is heavily biased with minimized variance, effectively becoming a pure Temporal Difference (TD) estimator. Conversely, at $\lambda = 1$, the estimator exhibits minimal bias but elevated variance, aligning with a pure Monte-Carlo approach.

GAE ensures a more stable and efficient learning trajectory for policy gradient methods in RL. By reducing the variance of the Advantage function, GAE facilitates more accurate policy updates, consequently enhancing performance across a range of tasks [32].

## 2.6 Temporal Difference Learning

Temporal Difference (TD) Learning is a pivotal position in RL [36]. Its versatility is evident in its applications, ranging from Value Function Updates to Policy Improvement, and many more. A distinguishing feature of TD methods is their ability to update estimates by bootstrapping (process of updating value estimates based on other, current value estimates) from other learned estimates, eliminating the need for a complete episode's final outcome. This allows TD to harness intermediate experiences to approximate the value function $V(s)$ without awaiting the culmination of the current episode. Consequently, TD methods can initiate updates immediately upon reaching the subsequent time-step. Such a mechanism renders TD learning particularly adept at handling scenarios with prolonged or indeterminate episode lengths, such as infinite time-series problems [36].

Upon reaching time-step $t+1$, TD methods can establish a dynamic target for the value function update. This target is then employed to generate updates using the reward $r_{t+1}$ and the projected return $V(s_{t+1})$. The TD error, a crucial metric in this context, is defined as [36]:

$$\delta = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{2.6}$$

Subsequently, the TD update for the value function $V(s)$ is articulated as:

$$V(s_t) \leftarrow V(s_t) + \alpha\delta, \tag{2.7}$$

Here, $V(s_t)$ and $V(s_{t+1})$ denote the value functions corresponding to states at time-steps $t$ and $t+1$, respectively. The parameter $\alpha$ represents the learning rate, $r_{t+1}$ signifies the ensuing time-step's reward, and $\gamma$ encapsulates the discount factor.

## 2.7 Preferential Temporal Difference Learning

Temporal Difference (TD) learning underpins many RL algorithms, offering a mechanism to approximate the value function of a policy. Despite its integral part of RL, traditional TD learning is not without shortcomings, especially when function approximation is utilized. Challenges emerge in areas like credit assignment between states and error propagation due to partial observability.

The Preferential Temporal Difference (PTD) Learning algorithm seeks to mitigate these limitations. Central to PTD is the preference function $\beta$, which for a given state, assigns a value in the range $[0, 1]$, signifying the state's relevance. Effectively, $\beta$ allows for the re-weighting of states in TD updates, equal with their perceived importance. For instance, when $\beta(s) = 0$, the corresponding state remains unchanged upon visitation, dropping its role in bootstrapping due to perceived irrelevance. Conversely, if $\beta(s) = 1$, the state undergoes full updates and contributes wholly to the bootstrapping process. However, this selective updating can incur its own complications; a state left unaltered might proffer inaccurate values, thereby introducing biased updates when employed as a target for states.

Addressing this challenge, Anand et al. [1] introduced the notion of beta returns, denoted $G_t^\beta$. This enables bootstrapping that's based by preference, which is captured by the following equation

$$G_t^B = r_t + \gamma \left[\beta(s_{t+1})V(s_{t+1}) + (1 - \beta(s_{t+1}))G_{t+1}^\beta\right]. \tag{2.8}$$

Equipped with this beta return, the PTD algorithm can be implemented using either a forward view (updating at every time-step) or a backward view (updating post-episode or after a defined number of time-steps). The latter is adopted in our

PTD implementation and respective baselines. In a linear approximation context, Anand et al. demonstrated that the expected updates steered by Preferential TD converge to a distinct fixed point.

The update for the value function using PTD is:

$$V(s_t) \leftarrow V(s_t) + \alpha(\beta(s_t)G_t^B + (1-\beta(s_t))V(s_t) - V(s_t)) \tag{2.9}$$

## 2.8 Actor-Critic Methods

Actor-Critic Methods belong to the family of Temporal Difference (TD) learning methods. Distinctively, they employ separate neural network architectures to represent both the policy and the value function, termed as the Actor and the Critic, respectively.

- **Actor**: This component is responsible for determining the agent's policy, denoted as $\pi(a|s;\theta)$. It prescribes actions, represented by $a$, based on specific states, symbolized by $s$. The parameter $\theta$ are the policy's parameters.

- **Critic**: This component evaluates the efficacy of the policy by estimating its corresponding value function, $V(s)$. The parameters governing this value function might be distinct from the policy's or could be shared.

These methods operate on an on-policy model, implying that the policy undergoes refinement throughout the learning process. The Critic leverages the TD error, as expressed in Eq(2.6), and updates its assessment using the TD update rule given by Eq(2.9) [31]. Following each action decision by the Actor, the Critic appraises the state to discover if the outcomes align with or strays from expectations. This assessment often employs the TD Error, as shown in Eq(2.6) [31]. A positive TD

error prompts the Critic to support for the repetition of the action in future scenarios. Conversely, a negative TD error suggests avoiding the repetition of the action [31].

Historically, many of early RL algorithms were rooted in actor-critic methodologies [47], [3], [31]. This demonstrates the pivotal role of this algorithm in the RL domain, serving as a foundation for advanced algorithms such as Advantage Actor Critic (A2C) [27], Asynchronous Advantage Actor Critic (A3C) [27], Proximal Policy Optimization (PPO) [33], and several others.

## 2.9 Advantage Actor Critic

The Advantage Actor-Critic (A2C) algorithm represents a prominent algorithm in RL. By synergizing the strengths of both value-based and policy-based strategies, A2C utilizes the Actor-Critic algorithm's actor and critic but introducing another component named the advantage.

Central to A2C's efficacy is the concept of 'advantage', quantifying the relative merit of executing action $a$ in state $s$ as compared against the average policy action. The advantage function, $A(s, a)$, takes shape this notion, and is defined as the discrepancy between the action-value function, $Q(s, a)$, and the state-value function, $V(s)$.

A noteworthy feature of A2C is its synchronous update. The agent accumulates experiences synchronously from multiple environment instances. It is found that A2C produces comparable performance to its asynchronous counterpart while being more efficient [27].

A2C's learning trajectory unfolds in two stages:

1. **Policy Evaluation**: Here, the critic appraises the current policy's value func-

tion, predicting the expected return for every state. This serves as a foundational baseline against which the actor's decisions are evaluated.

2. **Policy Improvement**: The actor, informed by the critic's insights, revises its policy parameters. It employs the advantage function as an indicator, calibrating the magnitude and direction of policy alterations.

Through the interplay between the actor and critic, A2C iteratively refines its policy, driving it ever closer to reward maximization. Such methodical, progress cements A2C's stature as a important algorithm in RL.

## 2.10 Proximal Policy Optimization

Traditional policy gradient methods enhance policies by orienting updates towards expected reward maximization, moderated by a learning rate. Yet, over-aggressive updates can occasionally lead to policy deterioration. Schulman et al. [33] introduced the Proximal Policy Optimization (PPO) algorithm to address this issue, emphasizing incremental policy updates to ensure stable learning.

PPO's main contribution is to moderate policy updates, ensuring they don't diverge significantly from a previous policy. To realize this, the objective function is appended with a clipping mechanism to limit drastic policy alterations.

The policy objective, defined as

$$L^{CLIP}(\theta) = \hat{E}_t \left[ \min \left( r_t(\theta) \hat{A}_t^{GAE}, \text{clip} \left( r_t(\theta), 1 - \epsilon, 1 + \epsilon \right) \hat{A}_t^{GAE} \right) \right], \tag{2.10}$$

with the following definitions:

- $\theta$ signifies the policy parameters.
- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the importance sampling ratio between the new ($\pi_\theta$) and old ($\pi_{\theta_{old}}$) policies.
- $\hat{A}_t^{GAE}$ is the advantage function's estimator through Generalized Advantage Estimation (GAE).
- $\epsilon$ is a hyperparameter describing the acceptable policy update magnitude.

The clipping function modifies substantial policy updates. If the revised policy varies extensively from its predecessor (exhibiting a high $r_t(\theta)$), it undergoes clipping, promoting moderation in optimization steps.

Moreover, to moderate the policy gradient estimates' variance, PPO incorporates a baseline value function, optimized through a mean squared error loss:

$$L^{VF}(\theta) = \min_\theta \sum_{n=1}^{N} (V_\theta(s_n) - \hat{V}_h)^2, \tag{2.11}$$

where $V_\theta(s_t)$ estimates state $s_t$'s value, parameterized by $\theta$, and $\hat{V}_h$ represents $s_t$'s target value.

The total PPO objective merges the clipped surrogate objective and value function loss:

$$L^{TOTAL}(\theta) = L^{CLIP}(\theta) - c_{VF} L^{VF}(\theta) + c_{Ent} S\left[\pi_\theta\right], \tag{2.12}$$

with $c_{VF}$ and $c_{Ent}$ as hyperparameters for the value function loss and an entropy regularization term $S[\pi_\theta]$ respectively. This entropy term bolsters exploratory behavior helping the agent finding optimal actions.

By embedding trust regions via clipping, PPO reduces each policy update's magnitude, ensuring closeness to the preceding policy and safeguarding against optimization divergences. PPO's state-of-the-art achievements across multiple RL challenges, combined with its elegance and efficiency, have made PPO a widely important RL algorithm.

## 2.11 Critical States & State Importance

In RL, understanding which states critically influence an agent's performance can be pivotal for effective learning and exploration. Karino et al. [22] introduced a method to quantify the significance of individual states, termed as state importance. This measure provides insights into a state's contribution to the overall task performance.

The essence of state importance lies in assessing the variability in the state-action values $Q^{\pi}(s, a)$, where the policy $\pi$ influences the state-action decision process. The variance is computed over all possible state-action pairs and is further adjusted based on the probability of selecting a specific action when the agent is in state $s$. Formally, the state importance for a state $s$ is defined as:

$$SI(s) = Var_{p(a|s)}[Q^{\pi}(s, a)], \tag{2.13}$$

where $Var_{p(a|s)}$ represents the variance across the action distribution for state $s$. A heightened state importance signifies a pronounced variability in the prospective rewards for different actions within that state, indicating its pivotal role in task performance. Such insights can serve as a heuristic for exploration strategies, directing the agent towards states of higher importance, thus potentially expediting the learning process.

However, a limitation of this methodology lies in its applicability. While algorithms that leverage state-action values can harness this technique to discern state importance, those relying solely on state values, $V^{\pi}(s)$, cannot utilize this approach.

To segregate states based on their significance, Karino et al. [22] employed state importance values, introducing a threshold that takes in consideration the top 10% of states as critical. This threshold is manually configured and does not have an autonomous procedure of selecting the percentile of states should be critical state for every environment.

Karino et al. [22] validated their methodology across various environments, both continuous and discrete in nature. Empirical results, particularly when integrating critical states within Deep Q-Networks for exploration or exploitation strategies, showcased the method's promising potential in bolstering RL algorithms.

## 2.12 Exponential Weighted Moving Average

Exponential Weighted Moving Average (EWMA) is a commonly used method for calculating a smoothed average of a time series. Given a set of input values $x_1, x_2, ..., x_n$ and a decay factor $\alpha \in [0, 1]$, the EWMA of the data is calculated recursively according to the following formula

$$EWMA_t = \alpha * x_t + (1 - \alpha) * EWMA_{t-1}. \tag{2.14}$$

In other words, the EWMA of the data is calculated by taking a weighted average of each value, where the weights decrease exponentially with time. The decay factor $\alpha$ determines the rate at which the weights decrease and controls the degree of smoothing in the resulting average. A higher value of $\alpha$ gives more weight to recent

values and results in a faster-decaying moving average, while a lower value of $\alpha$ gives more weight to past values and results in a slower-decaying moving average.

EWMA is often used in time series analysis to remove noise and highlight trends in the data. It is also commonly used in finance to calculate moving averages of stock prices, where it is known as the Exponential Moving Average (EWMA). One advantage of EWMA over other moving average methods is that it gives more weight to recent values, which can make it more responsive to changes in the data. However, the choice of the decay factor $\alpha$ can be critical, and different values may be more appropriate depending on the specific application.

One potential limitation of EWMA is that it can be sensitive to outliers in the data, which can skew the weighted average. In some cases, alternative smoothing methods such as median smoothing or robust smoothing may be more appropriate. However, EWMA remains a widely used and effective method for many time series applications.

## 2.13 Min-Max Normalization

Min-Max Normalization is a commonly used method for scaling and normalizing numerical data to a fixed range of values. Given a set of input values $x_1, x_2, ..., x_n(x_i)$ where $\min(x_1, x_2, ..., x_n) \neq \max(x_1, x_2, ..., x_n)$, Min-Max Normalization maps each value $x_i$ to a new value $y_i$ in the range $[0, 1]$ according to the following formula:

$$x_norm = \frac{x_i - \min x_i}{\max x_i - \min x_i} \tag{2.15}$$

In other words, Min-Max Normalization rescales the data to a new range based on the minimum and maximum values in the input set. This ensures that all values are scaled proportionally and preserves the distribution of the data. Min-Max Normal-

ization is often used in machine learning and data analysis tasks to standardize the input data, which can improve the performance of models that are sensitive to the scale of the input features.

One potential limitation of Min-Max Normalization is that it can be sensitive to outliers in the data, which can skew the scaling of the values. In some cases, alternative normalization methods such as z-score normalization or robust scaling may be more appropriate. However, Min-Max Normalization remains a useful and widely used method for many applications.

# Chapter 3

# Related Work

Anand et al. [1] presented a comprehensive and insightful study on PTD Learning. They evaluated their approach in grid-based tasks with fully observable and partially observable states. In the appendix of their paper, they have evaluated their works on the cart-pole problems to determine if their algorithm can be useful for fully observable environments. They used the algorithm Preferential Actor Critic, which is using preferential temporal difference networks with Actor-Critic architecture. The author's did not have an automatic way to calculate beta values and thus had to determine what threshold in the environment for the internal states. The beta values were determined based on thresholds for cart-pole velocity, angle, and position, assigning a static value of either 1 for states surpassing the threshold or 0.1 for those that did not (Table 4.2). However, this method is impractical for environments with non-trivial observation spaces (e.g., MuJoCo Environments [41]) or when states are represented as images, such as in Atari environments. Since the MuJuCo environments have multiple different internal observations, determining the optimal beta values is extremely difficult/impractical. Another problem with this approach is that it does not fully utilize the beta range [0,1] and instead uses static assignments to the beta values.

For example if the state passes the threshold compared to the previous beta state, it will have a beta value if 1, otherwise any other states will be a beta value of 0.1. This is not fully utilizing the beta state algorithm as some states could be weighed more than others.

A related family of algorithms is Emphatic Temporal Difference (ETD) [37] algorithms. These algorithms employ an interest function to re-weight updates. The work by Sutton, Mahmood, and White [37] presents a novel method to enhance the performance of parametric temporal-difference (TD) learning algorithms. The authors propose selectively emphasizing or de-emphasizing updates on different time steps. They demonstrate that varying the emphasis of linear TD($\lambda$)'s updates in a specific manner stabilizes its expected update under off-policy training. Specifically, ETD utilizes the interest of the previous state to generate emphasis at a given time. The updates are then modified based on the emphasis values, resulting in distinct emphasis updates for two different trajectories that converge at the same state. Although ETD and PTD share the idea of reweighing updates according to the agent's preference, ETD adopts a trajectory-based update method, while PTD employs a state-dependent parameter to adjust the updates.

Recent research by Martin et al. [24] highlighted the advantages of adopting an adaptive interest function in Emphatic Reinforcement Learning over a uniform interest across all states. This adaptive approach, which places emphasis on critical states such as bottlenecks, has shown promise in transfer learning applications. The authors underscored the challenges of crafting effective interest functions in intricate environments and advocated for an interest function that dynamically adapts based on the agent's interactions within the environment. Drawing inspiration from prior work on hyper-parameter discovery [50] [54], objective function design [49], intrinsic

rewards [55], and temporal abstraction [42], Martin et al. proposed an online meta-gradient-based approach to learn this adaptive interest function. Notably, both meta-parameters and the parameters of the policy and value function are updated concurrently through gradient descent.

The work by Chelu, Precup, and van Hasselt [9] explores the problem of credit assignment in RL. The authors investigate how an agent can optimally use additional computation to propagate new information by planning with internal models of the world to enhance its predictions. They examine the advantages and peculiarities of planning implemented as forethought using forward models or as hindsight using backward models. In various carefully designed scenarios, the authors determine the relative merits, limitations, and complementary properties of both planning mechanisms. The key distinction between the backward model and PTD is that PTD is model-free. The study works to understand the gains of planning using forethought with forward model or hindsight operating with backward models. The paper delves into the optimal use of models in planning, focusing mainly on selecting the states in which predictions should be re-evaluated. Finally, the authors discuss model estimation and present a spectrum of methods extending from environment dynamics predictors to planner-aware models.

Addressing the credit assignment challenge, Velu et al. [43] introduced the Hindsight DICE algorithm. Their research highlighted the limitations of prior hindsight policies, specifically the temporal delay between the observation of non-trivial rewards and individual steps [15], which often resulted in suboptimal learning in complex environments [43]. Velu et al. analyzed that importance-sampling ratio estimation techniques can significantly enhance the stability and efficiency of off-policy methods applied to hindsight policies. They adopted the Dual stationary DIstribution Correction

Estimation (DualDICE) approach [29] and integrated an adaptive importance ratio approximation technique for hindsight policies, creating the Hindsight Distribution Correction Estimation (H-DICE) [43]. Their empirical results demonstrate H-DICE's superiority over baseline methods, showcasing not only improved final rewards but also a more rapid convergence rate.

There are also other works that share the idea of ignoring updates on partially observable states like in [48], [40]. They use a trajectory-based value for the partially observable state as a substitute to the value. The problem with this algorithm is that trajectory-dependent values are poorly understood compared to Preferential Temporal Difference Learning which is shown to be unbiased and it can be understood from a theoretical point since Anand et al. has provided theory related to PTD. In Xu et al.s work, their main problem is to interporlate direct value estimates and project the values to the rewards and previously estimated values. Since the work of Nishanth et al [1] is to ignore states that are deemed non-important, their works uses previously estimated values to do this step.

Jiang et al [20] extended Emphatic Temporal Difference Networks to Deep Reinforcement learning algorithms. The paper "Emphatic Temporal Difference Learning for Deep RL Agents" tackles the instability issues encountered in Temporal Difference (TD) learning algorithms when combined with function approximation and off-policy sampling, a problem known as the "deadly triad". The authors propose an extension of the Emphatic Temporal Difference ($ETD(\lambda)$) algorithm to deep RL agents to address this. $ETD(\lambda)$ algorithm ensures convergence in the linear case by weighting the $TD(\lambda)$ updates, but applying $ETD(\lambda)$ naively to popular deep RL algorithms, which use forward view multi-step returns, results in poor performance. Hence, the authors derive new emphatic algorithms for use in these contexts. The paper explains

the background on forward view learning targets and $ETD(\lambda)$, then proceeds to adapt $ETD(\lambda)$ to the forward view. The authors introduce a new multi-step emphatic trace for n-step TD and discuss further algorithmic considerations, including extensions for variance reduction, for the V-trace value learning target, and for the actor-critic learning algorithms. The algorithms' performance is first evaluated on small diagnostic Markov Decision Processes (MDPs), and then on classic Atari video games. The proposed methods demonstrate noticeable benefits in small problems, highlighting the instability of TD methods. Additionally, they report the highest score to date for an RL agent without experience replay in the 200M frames on Atari games, demonstrating the effectiveness of their approach in complex, real-world tasks.

The paper "Deep Successor RL" by Tejas D. et al. [25] introduces a novel method called Deep Successor RL (DSR), which incorporates Successor Representations (SR) into an end-to-end deep RL framework. The SR methodology decomposes the value function into two components: a reward predictor and a successor map, which represent the expected future state occupancy from any given state and the scalar rewards for each state, respectively. The value function is computed by taking the inner product between the successor map and the reward weights. DSR exhibits increased sensitivity to distal reward changes due to the factorization of reward and world dynamics, and it can extract bottleneck states (subgoals) from successor maps trained under a random policy. DSR consists of two sub-components: a reward feature learning component which uses a deep neural network to predict intrinsic and extrinsic rewards and learn useful features from raw observations, and an SR component which estimates expected future feature occupancy conditioned on the current state. The value function can then be calculated as the dot product of these two factored representations. The authors tested their approach on two environments using raw pixel observations,

the MazeBase grid-world domains and the Doom game engine. They demonstrated empirical convergence results on several policy learning problems and sensitivity of the value estimator given distal reward changes. They also illustrated the capability of extracting plausible subgoals for hierarchical RL by performing normalized-cuts on the SR.

IMPALA (Importance Weighted Actor-Learner Architecture) [12] is a distributed RL agent designed to solve a large collection of tasks using a single set of parameters. This presents a challenge due to the increased amount of data and extended training time. The IMPALA agent is not only efficient in single-machine training but also scales to thousands of machines without sacrificing data efficiency or resource utilization. The IMPALA system achieves stable learning at high throughput by combining decoupled acting and learning with a novel off-policy correction method called V-trace. Unlike the popular A3C-based agents, IMPALA actors communicate trajectories of experience (sequences of states, actions, and rewards) to a centralized learner. This decoupled architecture allows IMPALA to achieve very high throughput.The system's scalability and V-trace method enable IMPALA to achieve high data throughput rates of 250,000 frames per second, making it over 30 times faster than single-machine A3C. Additionally, IMPALA is more data efficient than A3C-based agents and is more robust to hyperparameters and network architectures, allowing it to make better use of deeper neural networks. The effectiveness of IMPALA is demonstrated by training a single agent on multi-task problems using DMLab-30, a set of 30 tasks from the DeepMind Lab environment, and on all games in the Atari-57 set of tasks. The results show that IMPALA achieves better performance than previous agents with less data and exhibits positive transfer between tasks due to its multi-task approach.

The Dyna AI architecture [35] integrates learning, planning, and reactive execution,

with a focus on environments where an agent's actions have nondeterministic effects and the agent does not have complete knowledge about the impact of its actions. Key aspects of Dyna include trial-and-error learning of an optimal reactive policy, the learning of domain knowledge in the form of an action model, planning to find the optimal reactive policy given the action model, and reactive execution. The central principle is that planning is equivalent to 'trying things in your head' using an internal model of the world. Dyna extends RL to include a learned world model. The architecture follows a generic algorithm that involves: observing the world's state and reactively choosing an action, observing the resultant reward and new state, applying RL to this experience, updating the action model based on this experience, and then repeating these steps for hypothetical world states and actions to apply RL to hypothetical experiences. The architecture has its theoretical foundation in dynamic programming and its relationship to RL, temporal-difference learning, and AI methods for planning and search. Various studies have explored the use of action models with RL methods, and the integration of RL methods with concepts from other architectures for practical tasks. The Dyna architecture is versatile and not confined to any specific learning method.

Temporal abstraction offers an alternative approach to traditional decision-making in reinforcement learning. Instead of deciding on actions at each individual timestep, temporal abstraction allows for decisions that span multiple timesteps, facilitating higher-level reasoning and strategy planning. This becomes particularly advantageous in scenarios where certain sequences of actions recur frequently or in environments characterized by prolonged episodes.

Researchers at McGill University have proposed an alternative perspective that merges the problem of discovering options with that of learning options. They introduced a new option-critic architecture [2] capable of concurrently learning the internal

policies, termination conditions of options, and policy over options without the necessity of additional rewards or subgoals. The proposed model has been developed based on the policy gradient theorem, leading to a gradual learning process for intra-option policies and termination functions. It can work with both linear and non-linear function approximators, under discrete or continuous state and action spaces. Experimental results have shown that the proposed approach can learn meaningful temporally extended behaviors effectively, without requiring subgoals, extra rewards, demonstrations, multiple problems, or any other special accommodations. It is suggested that this method offers the first end-to-end approach for learning options that scales to very large domains with comparable efficiency.

The study investigates the modeling of temporally abstract actions in RL [39], referred to as "options." Traditionally, the learning and planning of options have been done through semi-Markov decision process (SMDP) methods, which necessitate executing an option to completion, thus treating it as a black box. This limitation confines SMDP methods to terminating courses of action and restricts their capability to learn about multiple options concurrently. This paper presents "intra-option" learning methods that overcome these constraints. In contrast to SMDP methods, intra-option methods learn about an option from small fragments of experience consistent with that option, without needing the option to be executed entirely. This is accomplished by leveraging the fact that the SMDP generated by the options is rooted in an underlying Markov Decision Process (MDP), thereby enabling learning about the effects of temporally extended actions through temporal difference methods. Intra-option methods, which are examples of off-policy learning methods, allow learning from experience within a single option. They can learn about the model of an option without executing it, as long as some consistent selections are made. Additionally,

these methods can be used to learn about several different options at the same time. The authors introduce intra-option learning algorithms for both learning the values of the options and their models. The computational experiments demonstrate the flexibility and speed improvements of these methods, which prove more efficient than traditional SMDP methods.

# Chapter 4

# Methodology

## 4.1 Methodology

To fully leverage the capabilities of our proposed algorithm, we incorporate three essential components. First, we compute the State Importance $SI(s)$, which is subsequently passed through the Exponentially Weighted Moving Average (EWMA) function to smooth the $SI(s)$ values. We then apply Min-Max Normalization to normalize these SI(s) values within the range [0, 1]. Next, we employ the Generalized Beta Advantage Estimation (GBAE) to generate beta advantages $\hat{A}_t^{GBAE(\gamma,\lambda)}$ where $\gamma = 0.99$ and $\lambda = 0.95$. These beta advantages are used to calculate the policy loss for the P3O algorithm. We then compute the value loss by using mean-squared error with beta-values, combining the policy loss and value loss with entropy to create the P3O loss. This loss is subsequently utilized to update the network. In the following sections, we provide an in-depth explanation of the core techniques employed in the P3O algorithm.

### 4.1.1 Automatically Calculating Beta States

We present a method for the automatic calculation of function $\beta(s)$, which consists of three main components: State Importance ($SI$) computation, Exponentially Weighted Moving Average (EWMA) smoothing, and Min-Max Normalization.

Intuitively, state Importance $SI$ is a measure of how important the current state $s$ is to the potential success and failure of the agent. We calculate the $SI$ as the product of the variance of probability distribution $p(a_t|s_t)$ and $V^{\pi}(s_t)$.

$$SI(s_t) = Var[p(a_t|s_t)] * |V^{\pi}(s_t)|, \tag{4.1}$$

where $V^{\pi}(s)$ represents the value function from the critic. Although this modification does not capture the full SI as state-values provide less information than state-action values (used in [22]), it is necessary for compatibility with on-policy algorithms like Proximal Policy Optimization (PPO) that do not use state-action values in their update.

Creating the state importance was inspired from Karino et al's [22] but instead of using the variance of the Q-values weighted by the probability of action given state, we use the variance of the probability action given state multiplied by state value. By using this modified version we can use state-values very similarly to Karino et al's [22] state importance but the key distinction is that the probability will determine the critical state. This means that actions that the agent determines to have the highest probability and will return higher reward are determined as importance states while actions that the agent determines that won't be that impactful with lower returns are less important. Karino et al's state importance is very useful in algorithms that have q-values as these values can have more information of what is important or not.

Since q-values have action-state values compared to our algorithm, they hold more information and thus could retrieve a better representation of what state importance is.

The resulting $SI$ values are then passed through the EWMA function [18]

$$x_t = \alpha * SI(s_t) + (1 - \alpha) * x_{t-1}. \tag{4.2}$$

Here, $x_t$ is the smoothed value at time $t$, and $\alpha$ is the smoothing factor. $\alpha$ determines the weight given to the current observation versus the previous smoothed value. By adjusting the hyperparameter alpha, the degree of smoothing in a time series can be controlled. A larger alpha value assigns greater weight to the most recent observation, leading to a decrease in smoothing. Conversely, a smaller alpha value gives more weight to the previously smoothed values, resulting in greater smoothing of the time series.

The smoothed SI values ($x_t$) are then normalized using the Min-Max Normalization algorithm, transforming them into the range [0, 1]. These normalized values represent the beta states $\beta(s)$ that are used in both the Generalized Beta Advantage Estimate (GBAE) Eq (4.3) and P3O algorithm.

### 4.1.2 Generalized Beta Advantage Estimation

We build GBAE upon GAE [32] by incorporating state preference function $\beta(s)$ into the GAE function Eq (2.5). This allows GBAE to assign different levels of importance to states according to the environment. Moreover, this incorporation allows GBAE to adaptively control the bias and variance of advantages for different states.

It is worth mentioning that the GAE/GBAE algorithms focuses on finding a balance

between bias and variance in the advantage estimation by combining the properties of Temporal Difference learning (TD-learning) and Monte Carlo (MC) methods [32]. In the algorithms, the parameter $\lambda$ controls the balance between bias and variance, with $\lambda = 0$ adapting TD properties and $\lambda = 1$ demonstrating MC properties (i.e the discounted return $G_t$ Eq (2.1)). However in GBAE, with $\lambda = 0$ the algorithm adapts TD properties and $\lambda = 1$ demonstrates $\beta$ return Eq (2.8) with baseline properties ($G_t^{\beta}$ - $V(s_t)$).

In GBAE, the state preference function $\beta(s_{t+1})$ is incorporated into the advantage estimation, as shown in the following equation

$$\hat{A}_t^{GBAE(\gamma,\lambda)} = \delta_t + (1 - \beta(s_{t+1}))(\gamma\lambda)A_{t+1}^{GBAE}, \tag{4.3}$$

where $\gamma$ is the discount factor, typically set to 0.99, and $\lambda$ is the steeper discount factor, constrained within $0 \leq \lambda \leq 1$; $\beta$ represents the preference function for the current state; $\delta$ denotes the advantage, defined as $r_t + \gamma V(s_{t+1}) - V(s_t)$; and $r_t$ represents the reward at each timestep $t$.

With this modification, GBAE provides more flexibility in controlling the bias and variance of advantages based on the importance of states in the environment. In cases where the state preference function assigns a high importance (e.g., $\beta(s) = 1$) to a particular state, the advantage for that state will exhibit a higher bias and lower variance. On the other hand, when the state preference function assigns a lower importance (e.g., $\beta(s) = 0$), the advantage will have lower bias and higher variance. This results in more targeted and efficient learning process. Furthermore, the incorporation of state preferences in GBAE has the potential to enhance the performance of policy optimization algorithms in complex environments. This will then be used with the modified Proximal Policy Optimization (P3O) to help the policy

| Num | Observation | Min | Max |
|---|---|---|---|
| 0 | Cart Position | -4.8 | 4.8 |
| 1 | Cart Velocity | -Inf | Inf |
| 2 | Pole Angle | ∼-0.418 rad | ∼0.418 rad |
| 3 | Pole Angular Velocity | -Inf | Inf |

Table 4.1: Observation details for the CartPole-v1 environment.

| Condition | Cart Position | Cart Velocity | Pole Angle |
|---|---|---|---|
| 1 | 0.5 | 0.5 | 0.05 |
| 2 | 1.0 | 1.0 | 0.1 |
| 3 | 0.3 | 0.3 | 0.03 |

Table 4.2: The different conditions Anand et al. [1] configured for the Cartpole problem.

update.

In a comparative analysis between Beta Returns and Generalized Beta Advantage Estimation (GBAE) shown in the Appendix (A.1), it is clear that both methods share similar computational properties. Specifically, when the parameter $\lambda$ in GBAE is set to 1, it demonstrates characteristics related to Beta Returns. A similar observation can be made when $\lambda$ is set to 0, as the computation is the same as the Temporal Difference (TD) advantage, represented by $\delta_t = r_t + \gamma v_{t+1} - v_t$. This resemblance highlights the flexibility of GBAE in adapting to different learning scenarios, depending on the chosen parameter values.

In summary, GBAE extends the capabilities of GAE by integrating state preferences into the advantage estimation process. This modification allows GBAE to adaptively control the bias and variance of advantages based on the importance of states in the environment, providing a more targeted and efficient learning process. The incorporation of state preferences in GBAE has the potential to enhance the performance of policy optimization algorithms in complex environments.

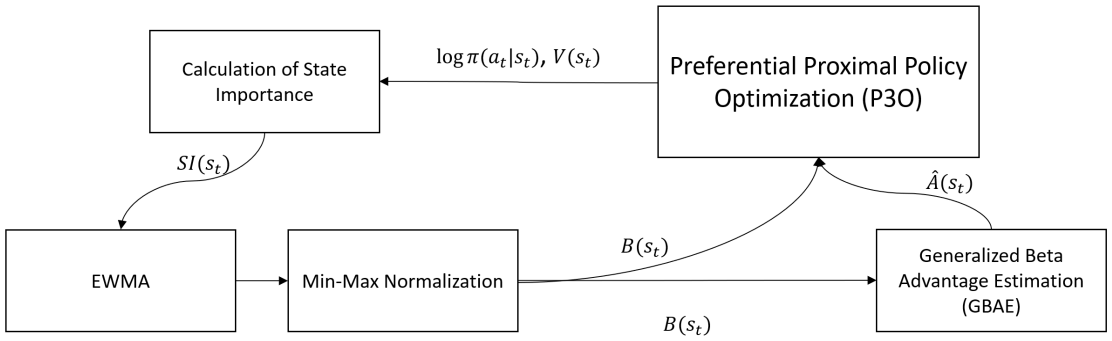



Figure 4.1: This figure represents the P3O (our implementation) pipeline. This demonstrates how the calculation of the beta values are utilized within the GBAE and P3O algorithm.

### 4.1.3 Preferential Proximal Policy Optimization

Preferential Proximal Policy Optimization (P3O) is a modified Proximal Policy Optimization (PPO) algorithm [33] that integrates the state-preference function $\beta(s)$ and GBAE algorithms (as the advantage estimator) to update its policy and value functions. While the policy structure remains identical to PPO, P3O replaces the advantage function estimator $\hat{A}_t^{GAE}$ with the GBAE-based advantage estimator $\hat{A}_t^{GBAE}$ in the policy loss computation. This change enables the P3O algorithm to prioritize important states over less important ones during updates. The policy update for P3O is defined as:

$$L_t^{CLIP}(\theta) = min(r_t(\theta)\hat{A}_t^{GBAE}, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t^{GBAE} \tag{4.4}$$

where $\theta$ denotes the policy parameters, $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the importance sampling ratio between the current policy $\pi_\theta$ and the old policy $\pi_{\theta_{old}}$, $\hat{A}_t^{GBAE(\gamma,\lambda)}$ is GBAE, $clip$ is the bounding mechanism that constraints the policy updates within a specific range, and $\epsilon$ is a hyper-parameter controlling the policy update size.

In P3O's policy, instead of having normal advantages where each advantage is

uniformly the same for bias and variance, it will have beta advantages that exhibit high-bias and low-variance if the state has high $\beta(s)$ values and vice-versa for low $\beta(s)$ values.

Furthermore, P3O introduces the automatic calculation of beta states $\beta(S)$, using method in section 4.1.1 . This approach allows the algorithm to automatically adapt the beta states based on the environment, overcoming the limitations of using static thresholds for determining beta states, as seen in previous works [1]. The following is the value loss for the P3O algorithm using mean-squared error

$$L_t^{VF}(\theta) = \sum_{n=1}^{N} \beta(s_n)(V_\theta(s_n) - \hat{V}_h)^2. \tag{4.5}$$

$V_\theta(s_t)$ is the predicted value of state $s_t$ by the value function parameterized by $\theta$, and $\hat{V}_h$ is the target value for state $s_t$ which we used as the discounted returns. The computation for the value function loss in P3O also differs from that in PPO.

We combine both the policy loss, value loss, and an entropy bonus $S$ like in PPO [33] to create the P3O objective function

$$L_t^{CLIP+VF+S}(\theta) = E_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)]. \tag{4.6}$$

where $L_t^{CLIP}$ represents the policy loss, $L_t^{VF}$ denotes the value loss, and $S[\pi_\theta]$ is the entropy bonus, which encourages the network to explore. The pseudo-code for P3O is a modification of OpenAI's Spinning Up documentation [30] and can be found in Algorithm 4.1.

In summary, P3O extends the original PPO algorithm by incorporating the state-preference function and GBAE to prioritize higher $SI$ during policy and value function updates. This modification enhances the algorithm's performance in complex environ-

**Algorithm 4.1** Preferential Proximal Policy Optimization (P3O)

**Input**: Initial policy/value function parameters $\theta_0$
**Output**: Final policy/value function parameters $\theta_N$
**for** k = 0, 1, 2, ...N-1 **do**
    Collect trajectories $D_k = \tau_i$ using policy $\pi_k = \pi(\theta_k)$ where $t$ represents timestep in trajectory $\tau_i$
    Compute rewards from environment ($r_t$)
    Compute method for Automatically Calculating Beta States $\beta(s)$ (section 4.1.1)
    Compute $\hat{A}_t^{GBAE}$ using GBAE with beta states $\beta(s_t)$ based on current value function Eq(4.3)
    Calculate policy loss using PPO-Clip objective and $\hat{A}_t^{GBAE}$ Eq(4.4)
    Calculate value function loss using mean-squared error Eq(4.5)
    Combine policy loss, value loss, and entropy bonus; update network Eq(4.6)
    Determine $\theta_{k+1}$ based on Eq(4.6) objective function
**end for**

ments and provides a more flexible approach to learning in situations with non-trivial observation spaces. A diagram of the algorithm can be found in Figure 4.1.

### 4.1.4 Design Reasoning

In our algorithm, we employ Generalized Advantage Estimation (GAE) due to its ability to address the high variance often seen in policy gradient methods. GAE offers an effective way to estimate the advantage function, as evidenced by its performance in environments like Biped and Quadruped from the MuJoCo suite [41], as demonstrated in [32]. We selected Proximal Policy Optimization (PPO) [33] as the foundation for our modifications, driven by its three key attributes: ease of tuning, straightforward implementation, and competitive performance compared to other leading methods.

Our proposed algorithm extends on the concept of state importance, using the Preferential Temporal Difference's (PTD) $\beta(s)$ to assign varying levels of significance to different states. Inspired by Karino et al.'s State Importance (SI) framework [22],

we seek a quantifiable measure of state importance. However, since our approach operates within an on-policy framework, which does not explicitly incorporate Q-values, we use a state-importance-inspired mechanism which exhibits similar properties.

We define the importance of a state based on two criteria: the potential for exceptionally high or low rewards, and the presence of a definitive action that leads to these rewards. Such states are considered important as they introduce a bias in the algorithm towards actions that lead to these extreme rewards. Conversely, states yielding uniform rewards, regardless of the agent's actions, are deemed unimportant.

To quantify this state importance, we use Eq (4.1), which essentially multiplies the variance of the action probabilities given the state, $p(a|s)$, with the state-value function, $v(s)$. This provides a scalar measure of state importance.

However, the determination of state importance is not an isolated event. To account for the effect of important states on their neighbors, we introduce an Exponentially Weighted Moving Average (EWMA) smoothing. This ensures that states preceding and succeeding an important state also get a boost in their importance scores, thereby guiding the agent's learning through a more effective sequence of updates. Neglecting this smoothing factor could impair the performance of the algorithm, particularly in complex environments such as MsPacman.

This smoothed state importance measure is versatile and adaptable, making it suitable for various types of environments, including those with intricate internal observations or image-based observations. It eliminates the need to predefine $\beta(s)$ conditions as proposed by the authors of Preferential Temporal Difference Learning [1] (Table 4.2). We therefore integrate this measure directly into our proposed Generalized Beta Advantage Estimation (GBAE) as defined in Eq 4.3, augmenting the PPO policy.

This modification to GBAE ensures that the advantages associated with high-importance states bear a higher bias and lower variance, thereby steering the algorithm towards these states. We found that this biasing mechanism enhances the performance of our algorithm, emphasizing the value of incorporating state preferences into the PPO policy optimization process.

# Chapter 5

# Experiments

## 5.1 Atari Environments

In this section, we detail the performance of our introduced Preferential Proximal Policy Optimization (P3O) method. We contrast its results with the standard Proximal Policy Optimization (PPO) [33] over six Atari 2600 games. We gauge the algorithms based on their average returns from ten different training runs (Fig. 5.5), individual episode returns (Fig. 5.1), variance (Fig. 5.7), entropy (Fig. 5.9), explained variance (Fig. 5.11), and both policy (Fig. 4.4) and value loss (Fig. 4.5). All results are post a training span of 20 million timesteps.

### 5.1.1 Training Details

In order to ensure a fair comparison between our proposed Preferential PPO (P3O) algorithm and the baseline PPO, we maintained identical hyperparameters for both algorithms. The code for the PPO and P3O algorithm was adapted from CleanRL [17]. Both models employed a learning rate of $1 \times 10^{-3}$, which yielded the best results across

all tested environments. The algorithms took 256 steps before updating, and each algorithm was trained concurrently in 256 environments. The discount factor $\gamma$ and the trace decay parameter $\lambda$ were set to 0.99 and 0.95, respectively. The number of minibatches and epochs were both set to 8. All environments were trained for a total of 20 million timesteps. The neural network architecture for both the actor and the critic consisted of three convolutional layers and two linear layers, with rectified linear units employed as activation functions. We trained both P3O and PPO algorithms on each of the 6 environments using 10 different random seeds and calculated the average returns for each run over specific episodes. This approach ensures a fair comparison between both algorithms, as RL algorithms are prone to high variance [7].

### 5.1.2 Atari 2600 Environments

We utilized a suite of six Atari 2600 environments [4] provided by OpenAI's Gym toolkit [8] to train and evaluate our algorithms. These challenging environments, requiring complex decision-making, serve as popular benchmarks for RL algorithms making it the golden standard to train and test our algorithm:

- **Breakout:** A classic arcade game (1976) where players control a paddle to bounce a ball and break bricks. The agent receives $+1$ reward point for breaking each brick and the game ends when all bricks are cleared or when the player loses all their balls. The observation space consists of a 210x160x3 RGB image which represents the game's screen.

- **Freeway:** Released in 1981, this game involves guiding a chicken to cross a ten-lane highway while avoiding being hit by cars. The player receives a reward of $+1$ point for each successful crossing and no penalty for being hit by a car.

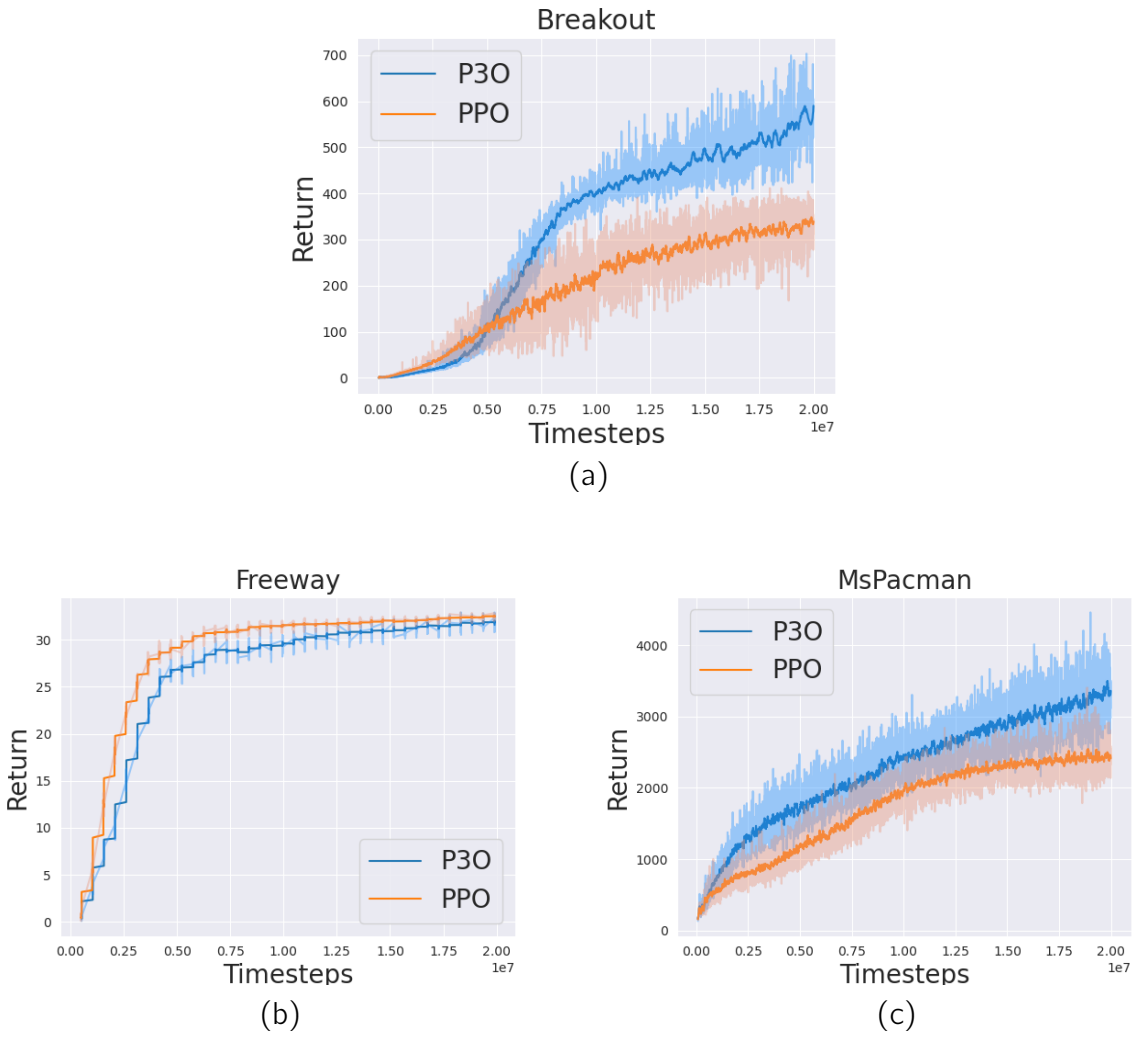
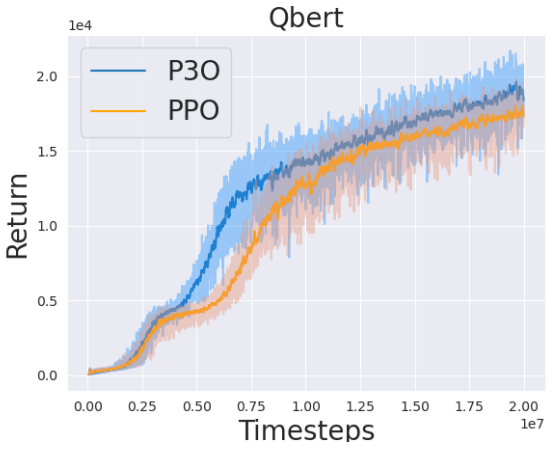
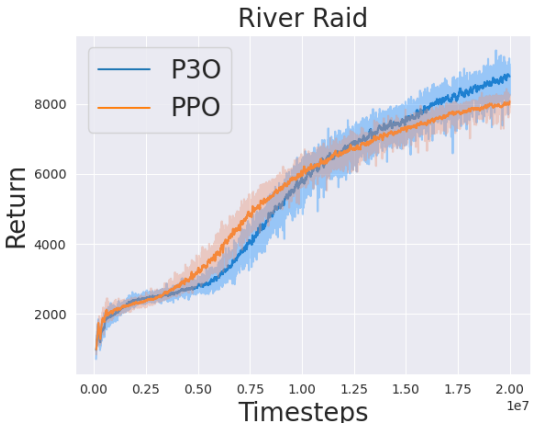


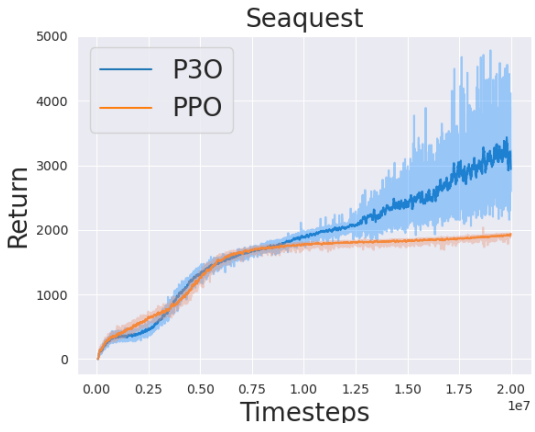

Figure 5.1: These figures represent returns of each environment where it takes 10 different runs at each episode and computes the average. This is different from taking the mean of the last ten episodes and plots what return the model provides for each episode. The blue line is the proposed algorithm (P3O) and the orange line is the baseline model (PPO).




(a)




(b)




(c)

Figure 5.2: These figures represent returns of each environment where it takes 10 different runs at each episode and computes the average. This is different from taking the mean of the last ten episodes and plots what return the model provides for each episode. The blue line is the proposed algorithm (P3O) and the orange line is the baseline model (PPO).

The episode ends after 2,016 time steps. The observation space, similar to Breakout, is a 210x160x3 RGB image.

- **Qbert:** A 1982 puzzle game where the character Qbert navigates a pyramid structure, changes cube colors, and avoids enemies. The agent receives +25 points by changing the color of the cubes to their target color, +300 points for catching a green ball or by luring an enemy into jumping off the pyramid, and +500 points for completing a level. The episode concludes when the player loses all lives. The observation space is a 210x160x3 RGB image.

- **River Raid:** A scrolling shooter game (1982) where the agent controls a jet flying over a vertically scrolling river. The agent is rewarded with points for destroying various types of enemies (ranging from +30 to +500), +100 points for collecting fuel, and an additional +500 points for reaching the end of a segment. The game concludes when the player crashes or runs out of fuel. The observation space is a 210x160x3 RGB image.

- **MsPacman:** In this 1982 game, the agent navigates Ms. Pac-Man through a maze, eating pellets and avoiding ghosts. The agent receives +10 points for each pellet or fruit eaten, and +200, +400, +800 and +1600 points for eating the first, second, third and fourth ghost respectively after eating a power pellet. The episode ends when the player loses all lives. The observation space is a 210x160x3 RGB image.

- **Seaquest:** A 1983 game where the agent controls a submarine to rescue divers, collect treasures, avoid or destroy enemy fish, and ensure sufficient oxygen supply by surfacing. The agent is rewarded +100 points for rescuing each diver, +200 points for every fish destroyed, +250 points for collecting treasure, and +500

points for surfacing when the oxygen level is low. The episode ends when the submarine is hit by an enemy or runs out of oxygen. The observation space is a 210x160x3 RGB image.

These diverse environments offer a strong assessment of the RL algorithms' performance and decision-making abilities while the reward structure in each environment encourages the development of efficient strategies and intelligent navigation.

### Atari Envpool Environments

Efficient exploration and learning from a large number of environments is a key challenge in RL. To address this issue, we employed CleanRL's Envpool code [46], which utilizes the Envpool Python library for creating and managing a pool of environments. This library enables the efficient execution of multiple environments in parallel, reducing the total time required for training. The Envpool package allows for parallelization of the training process across multiple CPU cores or GPUs. This significantly accelerates the training process and reduces the total time required for experimentation.

In summary, the Envpool package provides a powerful tool for efficiently exploring and learning from a large number of environments in RL. By leveraging the package's flexible interface and built-in tools, we can substantially improve the efficiency and scalability of our RL algorithms.

### Initialization of Networks

In our work, we used orthogonal initialization for both the policy and critic network. The motivation behind this decision is that orthogonal initialization tends to produce a more stable learning process in deep networks by preserving the variance of inputs through the layers [16]. This can prevent issues such as the vanishing or exploding

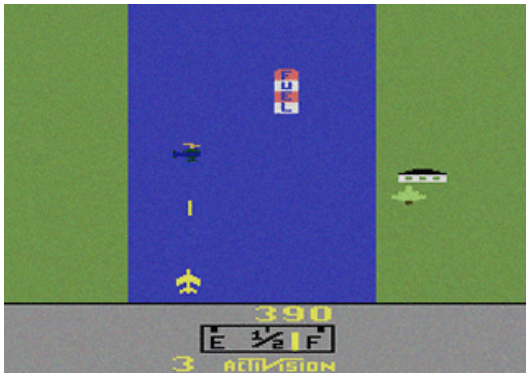
(a)

(b)


(c)

Figure 5.3: These are the games for each of the tested environments. We have tested on six different diverse environments with multiple different action spaces and objectives. For example Riverraid is vastly different to Freeway and Ms. Pacaman and vice versa. Having a diverse pool of different games can allow P3O to demonstrate how to can it perform on multiple unique environments.


(a)




(b)


(c)

Figure 5.4: These are the games for each of the tested environments. We have tested on six different diverse environments with multiple different action spaces and objectives. For example Riverraid is vastly different to Freeway and Ms. Pacaman and vice versa. Having a diverse pool of different games can allow P3O to demonstrate how to can it perform on multiple unique environments.

gradient problem. We found that this initialization strategy worked well in our context and helped to maintain a balanced importance across states in the policy and critic's learning process. We did experiment with other initialization strategies but found that orthogonal initialization provided the best results in our experiments.

### Returns and Variance

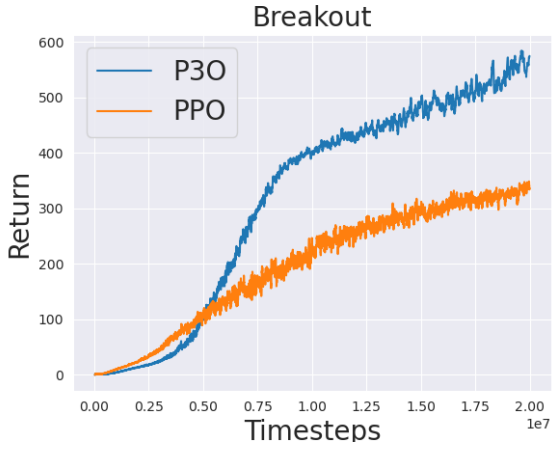
We trained both P3O and PPO algorithms on each of the 6 environments using 10 different random seeds and calculated the average returns for each run over specific episodes. This approach ensures a fair comparison between both algorithms, as RL algorithms are prone to high variance [7]. Our results indicate that P3O was able to achieve higher performance during the training phase, with P3O achieving superior returns in most environments. In the Breakout environment, P3O outperforms PPO significantly during training. Around the 5 million timesteps mark, the algorithm obtains higher rewards and learns a new strategy to generate substantial rewards. Fig. 5.5 for Breakout demonstrates that the final 10 episodes' average, across the 10 environments, reached a peak reward of 600. Examining Fig. 5.1, we observe that Breakout sometimes attains rewards around 700. It is possible that training the algorithm with more timesteps would lead to even greater rewards. Fig. 5.7 displays the variance of training runs, indicating that P3O maintains substantially lower variance up to 15 million timesteps in the breakout environment. This observation correlates with Fig. 5.1, where, starting from 15 million timesteps, there is a more significant increase in the variation of returns per episode. We believe the algorithm's enhanced performance towards the end of its training is due to its ability to discover more effective strategies or solutions, resulting in higher rewards.

For the Freeway environment, P3O achieves similar training returns as PPO. While

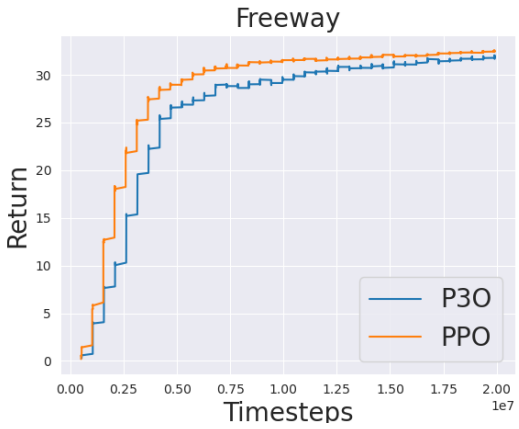
P3O is only slightly below PPO in terms of returns around 12.5 million timestep mark, it was still able to achieve exactly the same return at 20 million timestep. The difference is approximately 0.5 points between PPO and P3O. Comparing the variance of both P3O and PPO in Fig. 5.7, we find that they are quite similar.

In the environments of River Raid, MsPacman, and Qbert, P3O consistently outperforms PPO, achieving higher training returns. For the MsPacman environment, P3O demonstrates higher returns during training across 10 distinct environments, as depicted in Fig. 5.5 and Fig. 5.1. We adjusted the EWMA's alpha value to 0.75 during training, instead of 0.9 like in the rest of the environments (except for River Raid). In the River Raid environment, P3O managed to secure higher rewards than PPO, with approximately 1000 more returns. We set the EMWA value to 0.6 for this environment, deviating from the commonly used value of 0.9, leading to better performance. The variance of these environments—MsPacman, River Raid, and Qbert—are similar, as exhibited in Fig. 5.7. Near the conclusion of training for MsPacman and River Raid, P3O experiences a notable spike in variance across 10 different training runs. This anomaly arises from the algorithm yielding higher returns in certain episodes and lower returns in others. For example in River Raid environment, the algorithm is capable of reaping rewards up to 10,000-11,000 in most episodes but reverts to around 8,000 in others, hence the dramatic variance. In the Qbert environment, P3O maintains a lower variance throughout the training, with a spike occurring around the 17.5 million timesteps mark.

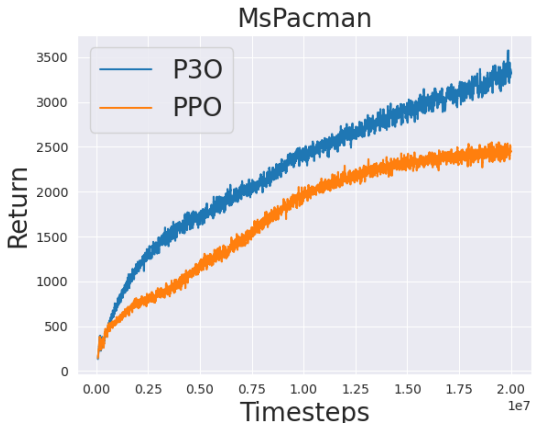
In the Seaquest environment, the training returns in Fig. 5.1 show that the algorithm begins to diverge from PPO around the 10 million timesteps mark. P3O rapidly attains higher returns at 10 million timesteps and quickly surpasses PPO. When comparing the returns per episode, the algorithm exhibits a larger returns variance. In




(a)

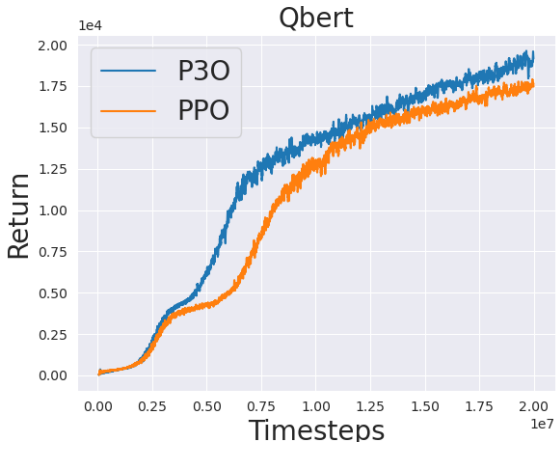



(b)

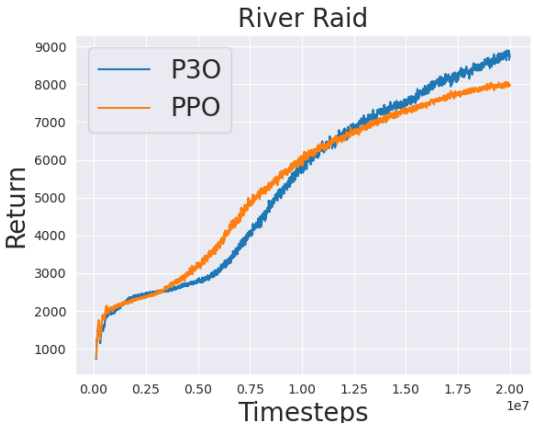



(c)

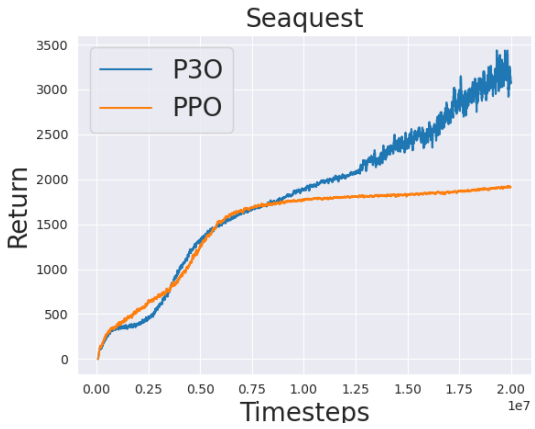
Figure 5.5: These figures represent the average returns of each environment where the blue line is the proposed algorithm (P3O) and the orange line is the baseline model (PPO). We first took the average of the last ten episodes and then calculated the average between 10 different runs at each episode. We then graphed the average of each episode to determine which model has better returns through training.




(a)




(b)




(c)

Figure 5.6: These figures represent the average returns of each environment where the blue line is the proposed algorithm (P3O) and the orange line is the baseline model (PPO). We first took the average of the last ten episodes and then calculated the average between 10 different runs at each episode. We then graphed the average of each episode to determine which model has better returns through training.

Fig. 5.1, the algorithm has a maximum return of greater than 4500 but a minimum return of greater than 2000 around the 20 million timesteps. This observation is evident in the variance Fig. 5.7, where, around the 15-20 million timesteps, the algorithm displays a considerably higher variance.

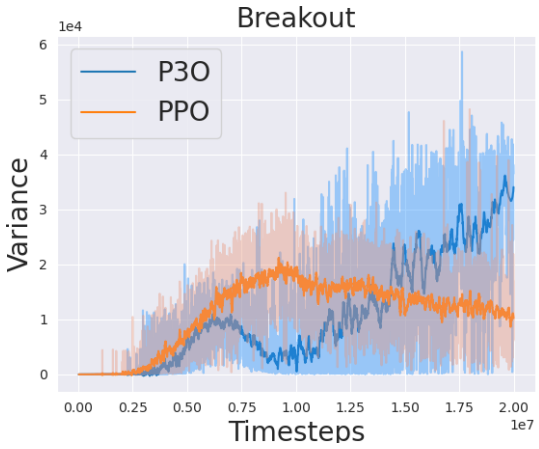
### Entropy and Explained Variance in Reinforcement Learning

Entropy and explained variance serve as key metrics to understand the behavior and learning efficacy of reinforcement learning (RL) algorithms.

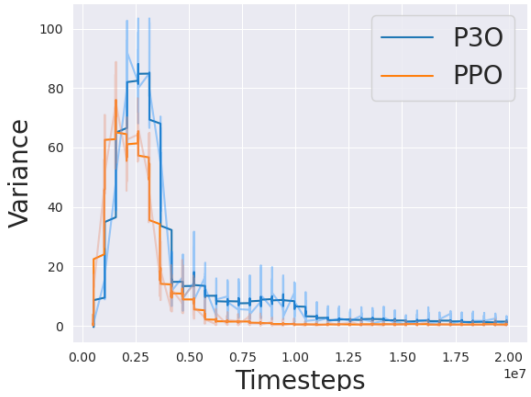
**Entropy** quantifies the uncertainty in the policy, shedding light on the algorithm's exploration-exploitation trade-off. A higher entropy suggests that the policy is more uncertain and thus leans towards exploration, sampling various actions. Conversely, lower entropy values indicate the agent is in an exploitation phase, with the algorithm relying on learned strategies. This balance between exploration and exploitation directly impacts an algorithm's ability to discover optimal policies.

**Explained Variance** gauges the predictive accuracy of the value function regarding received returns. A well-calibrated value function is paramount in RL, as it provides essential guidance during policy updates, especially in methods like P3O and PPO.

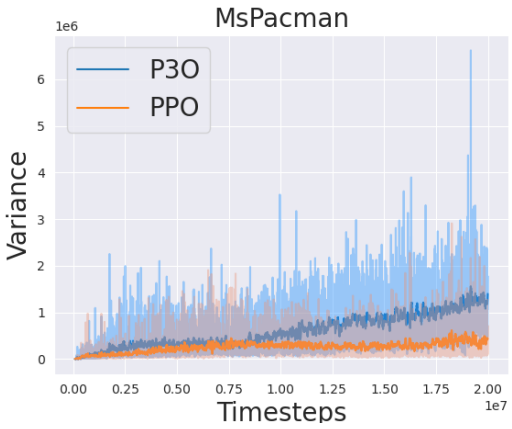
Examining Figure 5.9, P3O's entropy consistently registers lower values across environments, indicative of its tendency to exploit. This pronounced exploitation strategy is evident in the breakout environment, marked by a rapid entropy decrease. Such behavior suggests P3O's confidence in its learned strategy, eliminating the need for extensive exploration for the algorithm which is supported by superior returns observed in Figure 5.1 for breakout. For environments like Qbert, RiverRaid, and Seaquest, P3O commences exploitation earlier than PPO. This could stem from P3O's emphasis on significant states, potentially facilitating discovery of superior




(a)

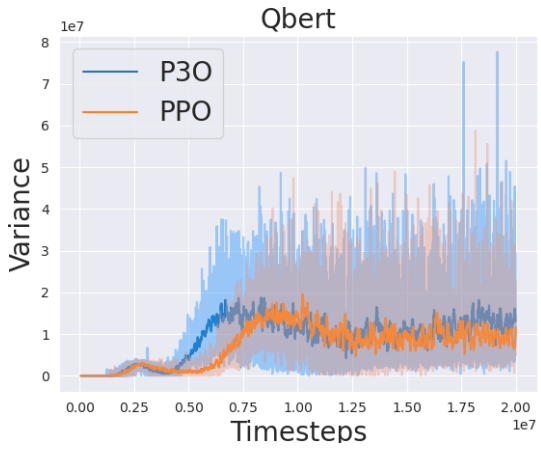



(b)

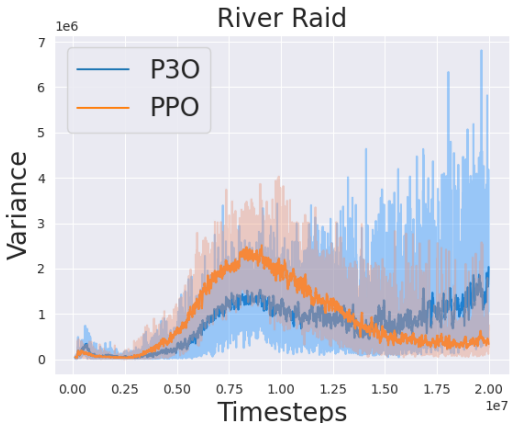



(c)

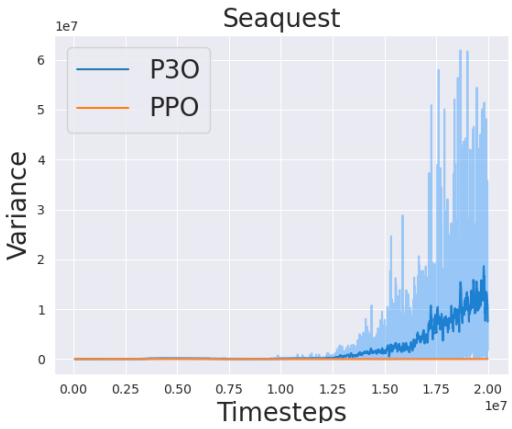
Figure 5.7: To quantify the variance of the proposed algorithm and baseline model, we computed the variance of their performance over 10 independent randomly seeded runs on each environment. Specifically, we retrieved the variance of the total rewards obtained at each episode and plotted the results on a graph. Lower values on the y-axis indicate less variance, which is desirable for reproducibility. The proposed algorithm is represented by the blue line, and the baseline model by the orange line. Each line has been smoothed so it can be read easily.




(a)




(b)




(c)

Figure 5.8: To quantify the variance of the proposed algorithm and baseline model, we computed the variance of their performance over 10 independent randomly seeded runs on each environment. Specifically, we retrieved the variance of the total rewards obtained at each episode and plotted the results on a graph. Lower values on the y-axis indicate less variance, which is desirable for reproducibility. The proposed algorithm is represented by the blue line, and the baseline model by the orange line. Each line has been smoothed so it can be read easily.

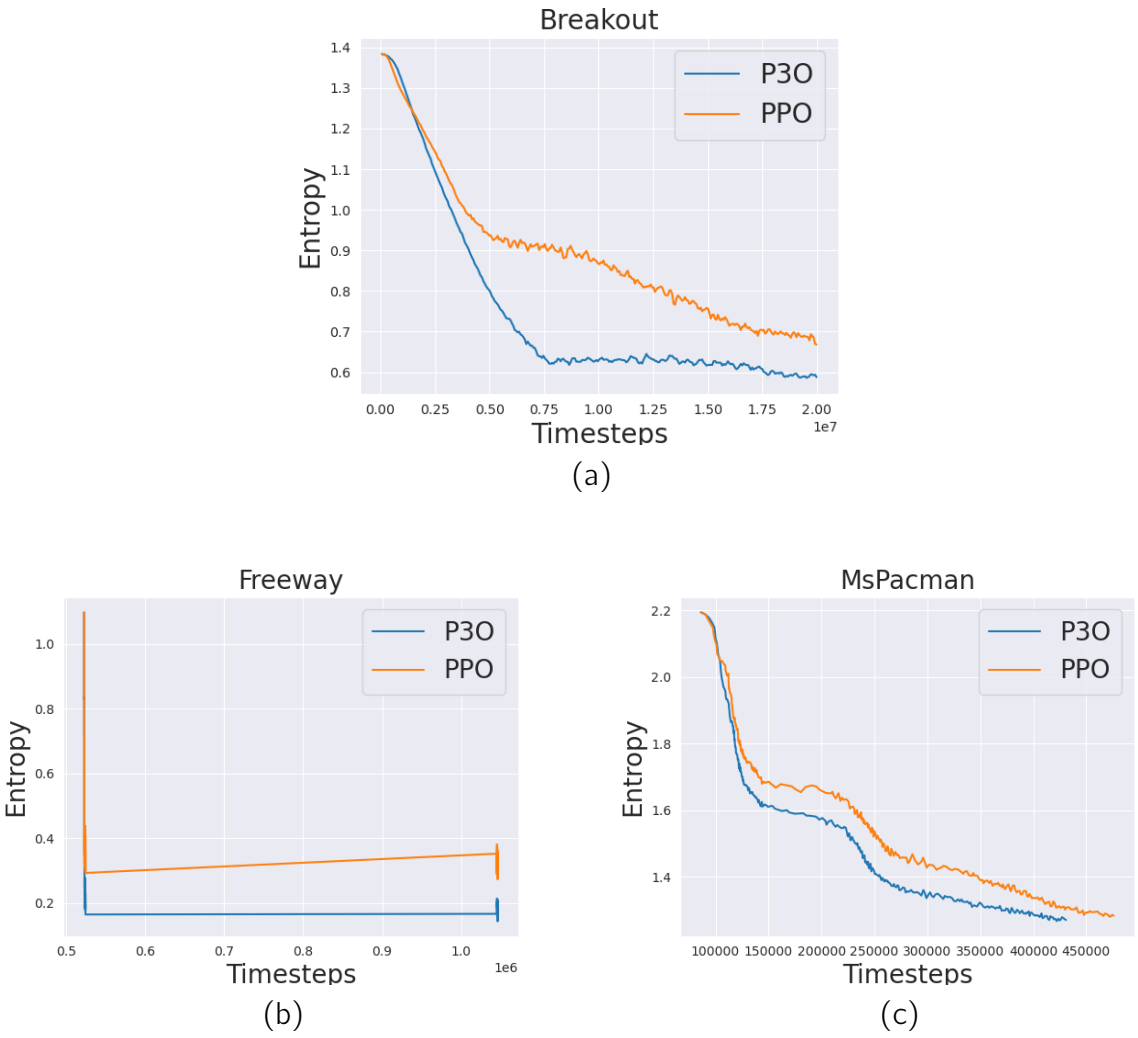
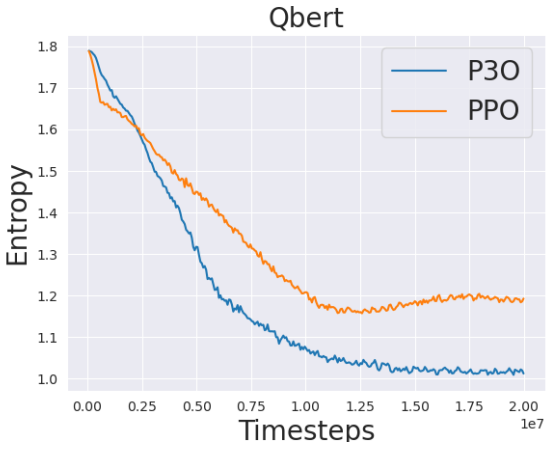
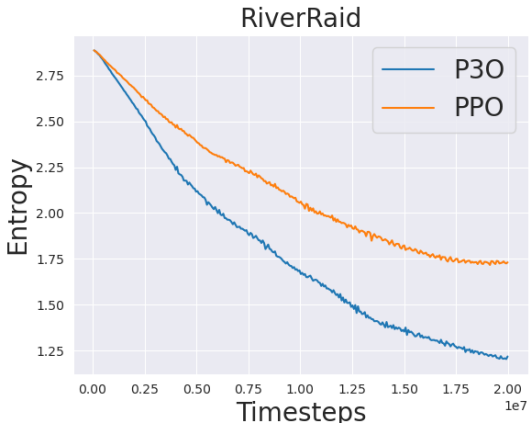


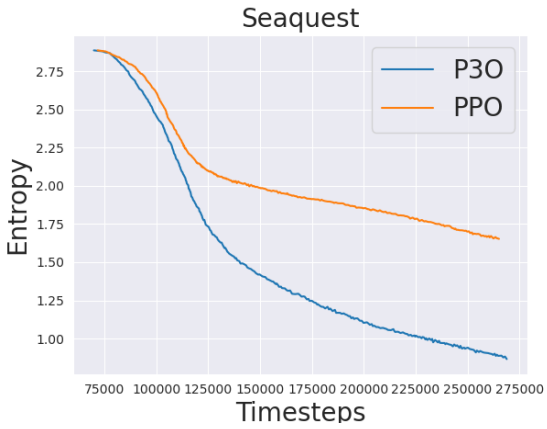

Figure 5.9: These figures represent the average entropy of each environment where the blue line is the proposed algorithm (P3O) and the orange line is the baseline model (PPO). Entropy measures the balance between exploration and exploitation in an algorithm. A higher entropy value indicates greater exploration, while a lower value suggests more exploitation.




(a)




(b)




(c)

Figure 5.10: These figures represent the average entropy of each environment where the blue line is the proposed algorithm (P3O) and the orange line is the baseline model (PPO). Entropy measures the balance between exploration and exploitation in an algorithm. A higher entropy value indicates greater exploration, while a lower value suggests more exploitation.

policies, thereby yielding improved returns. However, in MsPacman, both PPO and P3O manifest comparable entropy values.

Turning our attention to Figure 5.11, most models demonstrate proficiency in deciphering the underlying reward function, with P3O nearing perfection. Such adeptness implies accurate future return predictions, crucial for policy and value function updates. Noteworthy observations include P3O's accelerated reward function comprehension in the Breakout and Qbert environments, as compared to PPO. This enhanced understanding might account for P3O's out-performance over PPO in these scenarios.

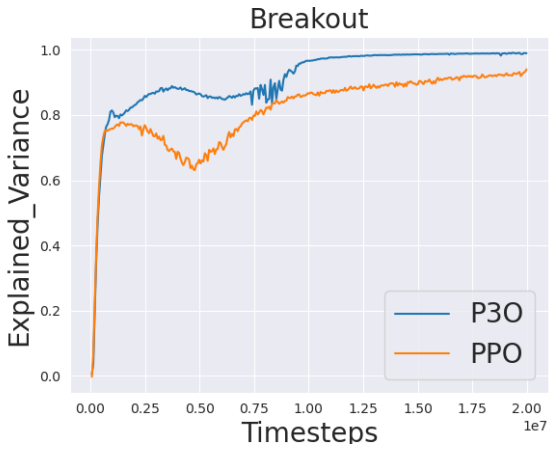
### Policy and Value Loss

Understanding the dynamics of policy and value loss in reinforcement learning algorithms provides crucial insights into their learning behavior and efficiency.

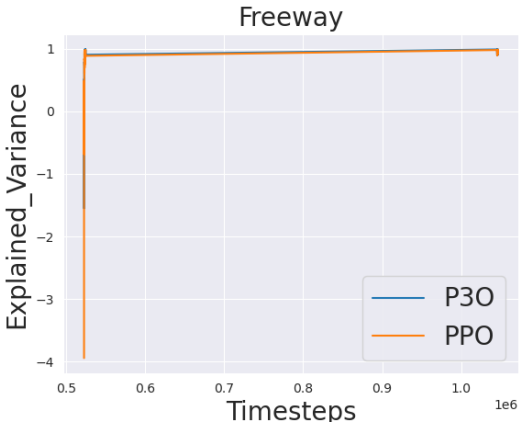
**Policy Loss** quantifies the alignment of the current policy with a desired or optimal behavior. Specifically, a decreasing, and ideally negative, policy loss suggests that the algorithm's current policy surpasses its preceding iteration in terms of aligning with the desired objectives. This typically indicates convergence towards an optimal strategy that aims to maximize rewards.

**Value Loss**, on the other hand, encapsulates the divergence between the predicted state values and the corresponding target values. These target values are typically derived from a discounted reward formula, incorporating potential future rewards. Minimizing this loss is paramount, as a reduced value loss directly translates to more accurate value predictions by the model. This, in turn, bolsters the agent's decision-making prowess, enabling it to discern and choose actions that maximize the estimated values of states.

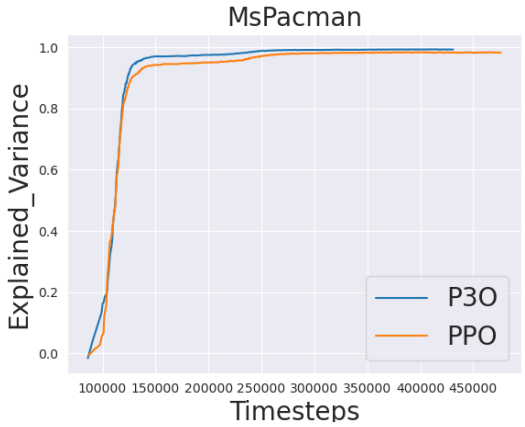
Examining Figure 5.13, P3O consistently discerns more optimal strategies, parti-




(a)




(b)




(c)

Figure 5.11: This graph demonstrates the average of the ten runs for each environment for explained variance. This graph demonstrates if the algorithms understand its reward function. The higher it is, its mostly likely understanding how to gain reward and if its lower, it means that the algorithm does not understand the reward function and its either getting lucky in getting reward or memorizing how to get reward. The blue line is the proposed algorithm (P3O) and the orange line is the baseline model (PPO).

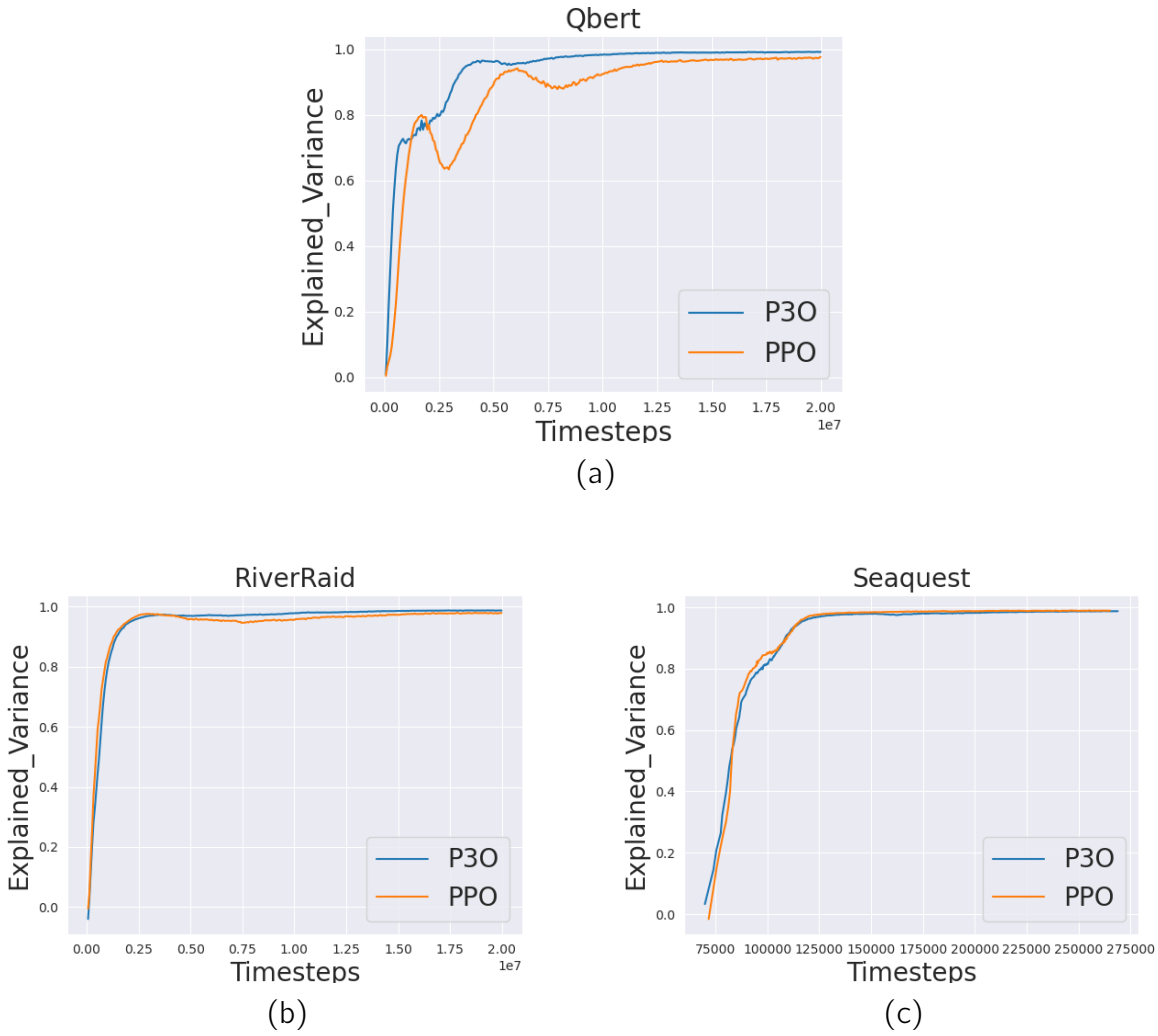
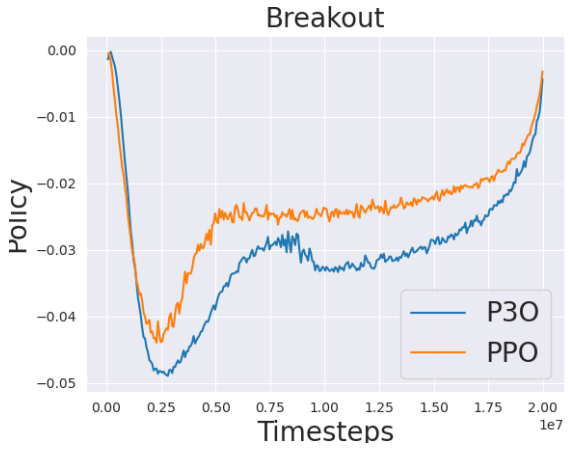
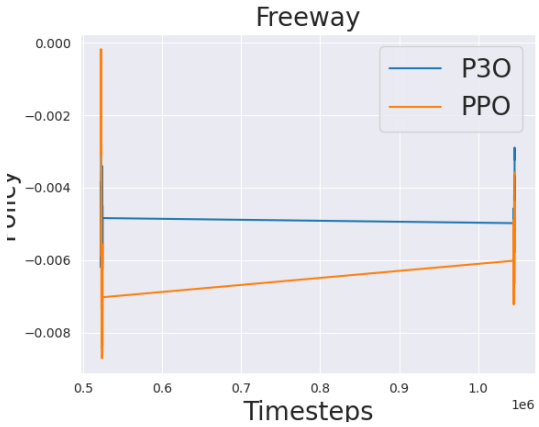


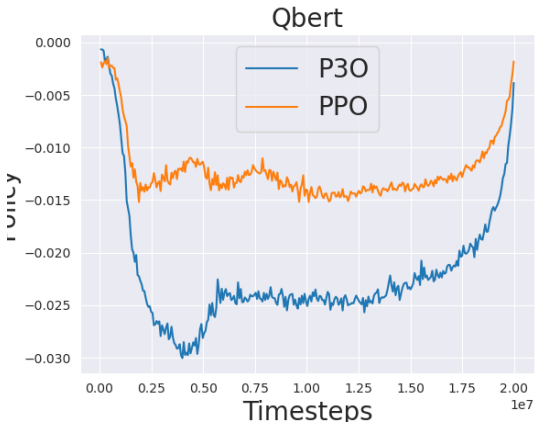

Figure 5.12: This graph demonstrates the average of the ten runs for each environment for explained variance. This graph demonstrates if the algorithms understand its reward function. The higher it is, its mostly likely understanding how to gain reward and if its lower, it means that the algorithm does not understand the reward function and its either getting lucky in getting reward or memorizing how to get reward. The blue line is the proposed algorithm (P3O) and the orange line is the baseline model (PPO).




(a)

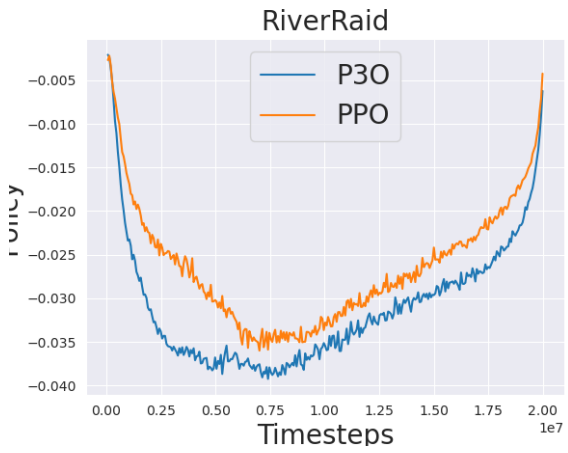



(b)

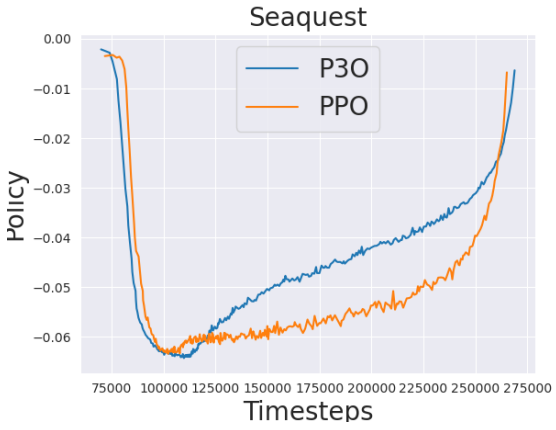



(c)

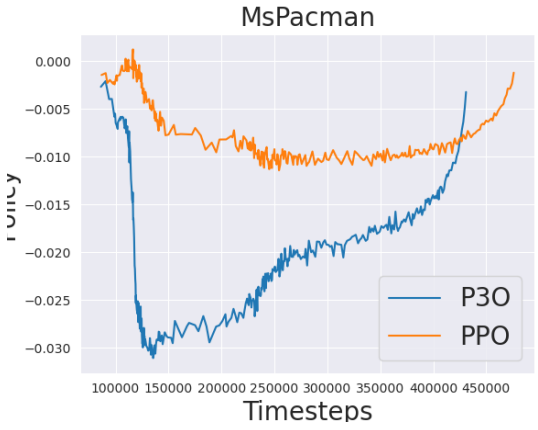
Figure 5.13: The policy shows the average update that it performs on both algorithms. This shows if the algorithm obeys the PPO properties where it doesn't update too much from the old policy. The blue line is the proposed algorithm (P3O) and the orange line is the baseline model (PPO).




(a)




(b)




(c)

Figure 5.14: The policy shows the average update that it performs on both algorithms. This shows if the algorithm obeys the PPO properties where it doesn't update too much from the old policy. The blue line is the proposed algorithm (P3O) and the orange line is the baseline model (PPO).

cularly around the midpoint of its iterations—RiverRaid and Seaquest being exceptions. This indicates P3O's general superiority in policy optimization relative to PPO. Notably, despite Seaquest presenting a higher policy loss for P3O, it still managed to outperform PPO. This anomaly warrants further investigation. On observing environments like Breakout, Qbert, River Raid and MsPacman, P3O's pronounced policy advantage becomes apparent.

Turning to Figure 5.16, PPO registers a pronounced value loss in comparison to P3O. This can be attributed to P3O's core design, wherein value updates are modulated by the beta values. Specifically, a diminished beta reduces the extent of value updates, whereas a heightened beta facilitates complete updates. This design choice implies that P3O's value loss will inherently be subdued for specific states in comparison to PPO. An intriguing exception emerges in the Seaquest environment, where P3O experiences an uptick in value loss towards the latter stages of its training, potentially resulting from sporadic instances of exceptionally high rewards.

### 5.1.3 Testing Results

In order to assess the performance of P3O and PPO with a trained model, we trained a single model for each algorithm on every environment using a randomly initialized seed. Subsequently, we tested these models on 10 different randomly seeded environments. Our results can be found in Table 5.1.

In the Breakout environment, the P3O model exhibited superior performance during testing. Upon retrieving 10 different runs from the P3O model, the algorithm achieved an average reward of 668.6, which is approximately 300 average reward points higher than that of PPO.

Significant improvements were also observed in the Seaquest, Qbert, and MsPac-

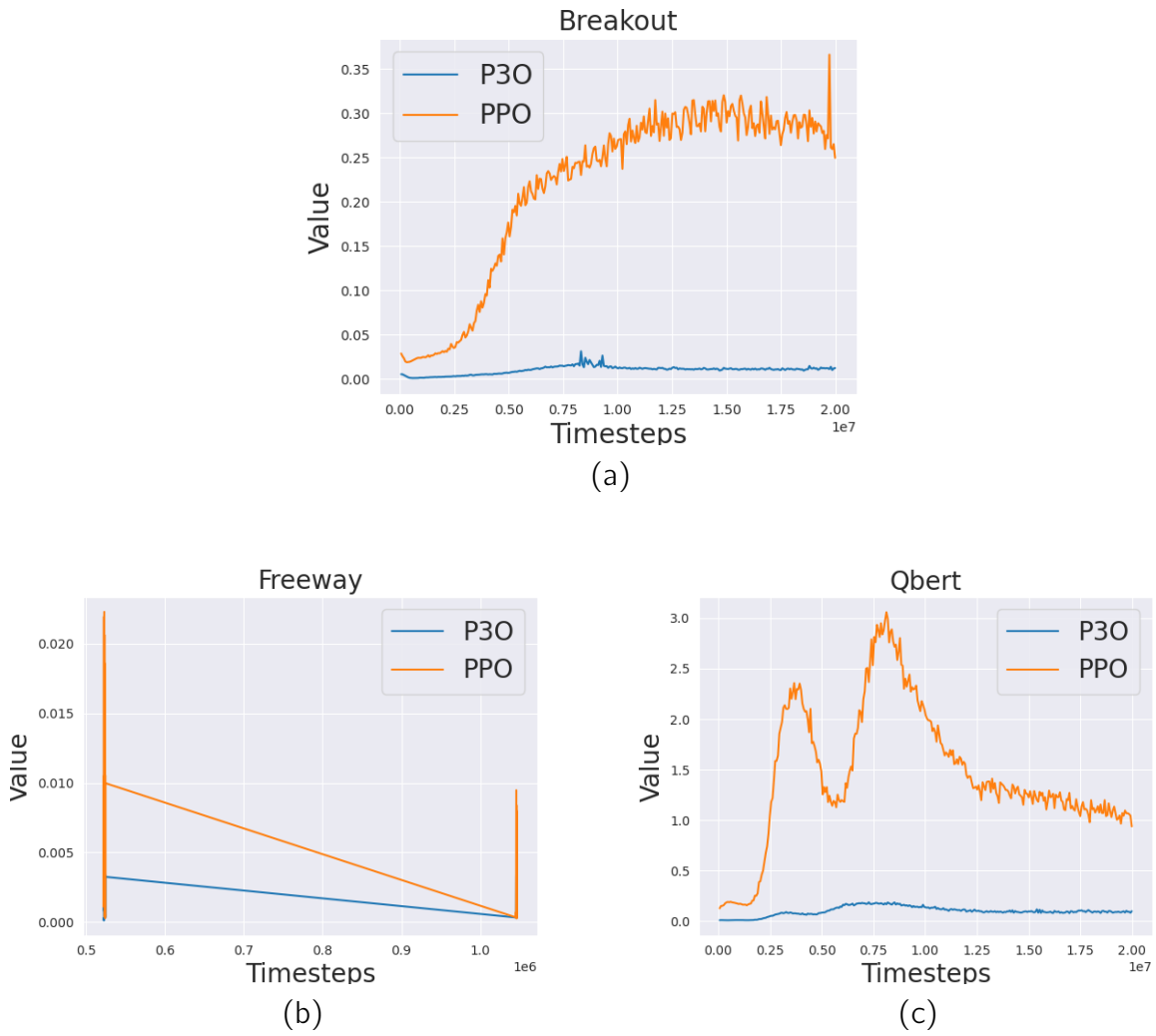
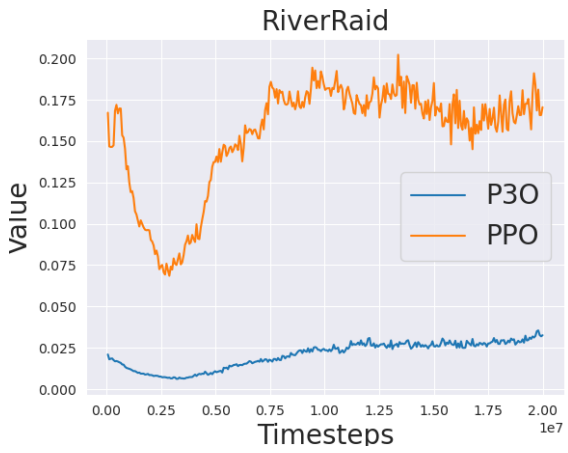
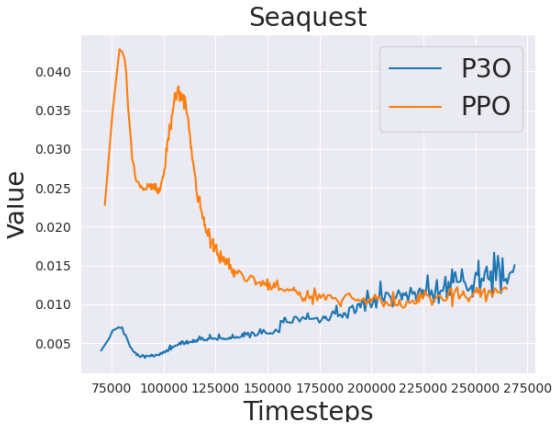


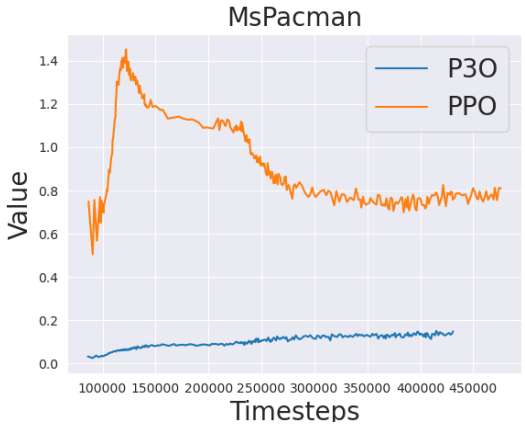

Figure 5.15: This graph is the average value that the network predicts for 10 runs in 6 different environments. This will show what the update for the value network will be and can determine whether appropriate updates for each is better. The blue line is the proposed algorithm (P3O) and the orange line is the baseline model (PPO).




(a)




(b)




(c)

Figure 5.16: This graph is the average value that the network predicts for 10 runs in 6 different environments. This will show what the update for the value network will be and can determine whether appropriate updates for each is better. The blue line is the proposed algorithm (P3O) and the orange line is the baseline model (PPO).

man environments. In the Seaquest environment, the P3O algorithm demonstrated an increase of approximately 800 average rewards. For Qbert, the algorithm displayed an increase of around 2000 average rewards. One of the most pronounced gaps between P3O and PPO average returns was observed in the MsPacman environment. While PPO attained an average reward of approximately 2000, P3O reached an average reward of 5015.0, indicating an increase of nearly 3000 rewards. Conversely, PPO outperformed P3O in terms of average rewards in the Freeway environment.

The experimental results obtained from our study suggest that the proposed P3O algorithm demonstrates potential as a RL algorithm for Atari environments as it outperforms the baseline in several environments. P3O's capacity to learn varying beta values based on SI enables the comprehensive utilization of the beta value range between [0,1], thereby enhancing the algorithm's overall performance. The algorithm exhibits remarkable performance in environments such as Breakout, Qbert, MsPacman, and Seaquest, where the significance of specific states is particularly crucial for obtaining high rewards. Our analysis reveals that P3O emphasizes states with high SI that are essential for its updating and bootstrapping, which accounts for its improved performance compared to PPO.

Nonetheless, our findings also imply that the efficacy of P3O is dependent upon the environment. For example, Freeway environment did not display improved performance with P3O. The variance of the P3O algorithm and the baseline PPO algorithm is generally comparable, with exceptions such as Breakout, Seaquest, and Qbert, where the variance is higher. In these environments, the increase in rewards is more substantial, and the algorithm's ability to explore various states and identify those with high SI is critical. For instance, in Breakout, the algorithm may have discovered a novel scoring method from these high-importance states, resulting in a significant

reward increase. We observed that the variance in Qbert escalated towards the end of training, indicating the necessity for additional training steps to stabilize the algorithm.

In conclusion, our experimental results indicate that P3O is a promising RL algorithm for Atari environments, with improved performance in specific environments. The proposed algorithm's capacity to learn varying beta values based on SI highlights its potential to enhance overall performance compared to the baseline PPO algorithm.

| | Models | |
|---|---|---|
| Atari Task | P3O | PPO |
| Breakout | **668.6** ±60.9 | 363.4 ±25.1 |
| Freeway | 22.9 ±0.6 | 22.8 ±0.5 |
| Seaquest | **2552.5** ±11.6 | 1806.0 ±6.4 |
| River Raid | **8898.0** ±88.4 | 8096.7 ±107.2 |
| Qbert | **18462.5** ±770.4 | 16855.0 ±751.8 |
| MsPacman | **5015.0** ±156.9 | 1975.0 ±77.1 |

Table 5.1: This table presents the performance results of two RL models: PPO and P3O. The models were trained and evaluated on a range of different environments, with the average scores computed across 10 independently randomly seeded runs for each environment. The scores reported in the table represent the mean of the 10 trails for each model and environment, along with the corresponding standard error.

| Hyperparameter | Value |
|---|---|
| Total Timesteps | 20 Million |
| Learning Rate | 1e-3 |
| Num. Environments | 256 |
| Num. Steps | 256 |
| Gamma | 0.99 |
| GAE parameter (lambda) | 0.95 |
| Minibatch Size | 8 |
| Num. Epochs | 8 |
| Clipping parameter epsilon | 0.1 |
| VF coeff. c1 (Eq.9) | 1 |
| Entropy coeff. c2 (Eq.9) | 0.01 |

Table 5.2: PPO and P3O hyperparameters used for training all six Atari environments. They were selected based on CleanRL's implementation of Proximal Policy Optimization [17].

# Chapter 6

# Conclusions

## 6.1 Contributions

In this study, we introduce a novel algorithm, Preferential Proximal Policy Optimization (P3O), which modifies the Proximal Policy Optimization (PPO) algorithm. P3O employs beta values calculated using State Importance SI, Min-Max Normalization, Exponentially Weighted Moving Average (EWMA), and beta advantages derived from the Generalized Beta Advantage Estimation (GBAE) algorithm. We evaluate the performance of the P3O algorithm in comparison to the baseline PPO algorithm across 6 Atari environments and conduct 10 distinct runs for each environment to account for variability in training returns.

Our results reveal that P3O outperforms PPO in 5 out of the 6 environments, specifically Breakout, MsPacman, Qbert, River Raid and Seaquest. We observe that the variance between the two algorithms is generally similar, with the exception of Breakout, River Raid and Seaquest, where the algorithms discover more effective methods for reward acquisition and consequently exhibit higher variance towards the end of training.

Entropy and explained variance metrics revealed that P3O tends to exploit more than PPO, indicating its confidence in learned strategies. Additionally, understanding policy and value loss is essential for gauging the efficiency of reinforcement learning algorithms. P3O generally demonstrated a better refinement for policy optimization and lower value loss compared to PPO, though there were some exceptions that merit further investigation. Our findings demonstrate the potential that Preferential Temporal Difference Learning can have on RL algorithms shown by P3O.

## 6.2 Limitations

Our algorithm demonstrated promising results. However, the introduction of an additional hyper-parameter, the alpha value in EWMA, 2.14, may require further adjustments for different environments. Hyperparameter tuning is notoriously challenging, often requiring intricate strategies to identify optimal values [11]. Various methods, such as Bayesian Optimization [21] and Tree Parzen Estimations [5], have been proposed for this purpose [53]. Our goal is to determine the best hyper-parameter for each environment, thereby fully demonstrating P3O's potential. It's noteworthy that our current choice of the EWMA value was based on a trial-and-error approach.

## 6.3 Future Direction

For future research, we propose assessing P3O in Mujoco environments [41], which are renowned for their complexity and numerous internal observations. P3O could streamline the search for optimal beta values by utilizing SI with Min-Max normalization and EWMA and potentially achieve high rewards in some of Mujuco's environments. Additionally, we recommend testing the algorithm in partially observable

environments, as P3O's employment of SI could result in enhanced performance by allocating greater attention to states that yield high importance. Finally, we suggest investigating the application of the proposed beta values $\beta(s)$ within the Actor-Critic with Experience Replay (ACER) algorithm [45].

Another direction that we would explore is to test on real-world applications especially with applications with control. PPO is a very famous and easy to use algorithm that is used in many research and applications. To test and compare P3O with PPO could potentially showcase the usefulness of the algorithm in real-world applications. Some of these applications could be in robotics [19], stock trading [51], health care [52] and many more. Investing in more useful cases for P3O could be beneficial and help achieve better results in those applications especially autonomous driving applications [23] as we believe that our work could potentially excel in that application.

In our future work, we aim to extend our approach to integrate beta values directly into the action-value function, $Q(s, a)$, rather than limiting them to the state-value function, $V(s)$. While our current investigations have primarily focused on the application of PTD algorithms to $V(s)$, we believe that a modification for $Q(s, a)$ could broaden the applicability across a more diverse set of reinforcement learning scenarios. Such an extension holds promise for enhancing the performance of sophisticated off-policy algorithms, such as Twin Delayed DDPG (TD3) [13] and Soft Actor-Critic (SAC) [14].

# Bibliography


[1] Anand, N., and Precup, D. Preferential temporal difference learning. *arXiv preprint arXiv:2106.06508* (2021).

[2] Bacon, P.-L., Harb, J., and Precup, D. The option-critic architecture. In *Proceedings of the AAAI conference on artificial intelligence* (2017), vol. 31.

[3] Barto, A. G., Sutton, R. S., and Anderson, C. W. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, 5 (1983), 834–846.

[4] Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research 47* (2013), 253–279.

[5] Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems 24* (2011).

[6] Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680* (2019).

[7] Bjorck, J., Gomes, C. P., and Weinberger, K. Q. Is high variance unavoidable in rl? a case study in continuous control. *arXiv preprint arXiv:2110.11222* (2021).

[8] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[9] Chelu, V., Precup, D., and van Hasselt, H. P. Forethought and hindsight in credit assignment. *Advances in Neural Information Processing Systems 33* (2020), 2270–2281.

[10] Cobbe, K. W., Hilton, J., Klimov, O., and Schulman, J. Phasic policy gradient. In *International Conference on Machine Learning* (2021), PMLR, pp. 2020–2027.

[11] Cooper, A. F., Lu, Y., Forde, J., and De Sa, C. M. Hyperparameter optimization is deceiving us, and how to stop it. *Advances in Neural Information Processing Systems 34* (2021), 3081–3095.

[12] Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning* (2018), PMLR, pp. 1407–1416.

[13] Fujimoto, S., Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In *International conference on machine learning* (2018), PMLR, pp. 1587–1596.

[14] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905* (2018).

[15] Harutyunyan, A., Dabney, W., Mesnard, T., Gheshlaghi Azar, M., Piot, B., Heess, N., van Hasselt, H. P., Wayne, G., Singh, S., Precup, D., et al. Hindsight credit assignment. *Advances in neural information processing systems 32* (2019).

[16] Hu, W., Xiao, L., and Pennington, J. Provable benefit of orthogonal initialization in optimizing deep linear networks. *arXiv preprint arXiv:2001.05992* (2020).

[17] Huang, S., Dossa, R. F. J., Ye, C., Braga, J., Chakraborty, D., Mehta, K., and Araújo, J. G. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research 23*, 274 (2022), 1–18.

[18] Hunter, J. S. The exponentially weighted moving average. *Journal of quality technology 18*, 4 (1986), 203–210.

[19] Ibarz, J., Tan, J., Finn, C., Kalakrishnan, M., Pastor, P., and Levine, S. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research 40*, 4-5 (2021), 698–721.

[20] Jiang, R., Zahavy, T., Xu, Z., White, A., Hessel, M., Blundell, C., and Van Hasselt, H. Emphatic algorithms for deep reinforcement learning. In *International Conference on Machine Learning* (2021), PMLR, pp. 5023–5033.

[21] Jones, D. R., Schonlau, M., and Welch, W. J. Efficient global optimization of expensive black-box functions. *Journal of Global optimization 13* (1998), 455–492.

[22] Karino, I., Ohmura, Y., and Kuniyoshi, Y. Identifying critical states by the action-based variance of expected return. In *Artificial Neural Networks and*

*Machine Learning–ICANN 2020: 29th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 15–18, 2020, Proceedings, Part I 29* (2020), Springer, pp. 366–378.

[23] Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Al Sallab, A. A., Yogamani, S., and Pérez, P. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems 23*, 6 (2021), 4909–4926.

[24] Klissarov, M., Fakoor, R., Mueller, J. W., Asadi, K., Kim, T., and Smola, A. J. Adaptive interest for emphatic reinforcement learning. *Advances in Neural Information Processing Systems 35* (2022), 95–108.

[25] Kulkarni, T. D., Saeedi, A., Gautam, S., and Gershman, S. J. Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396* (2016).

[26] Liu, F., Tang, R., Li, X., Zhang, W., Ye, Y., Chen, H., Guo, H., and Zhang, Y. Deep reinforcement learning based recommendation with explicit user-item interactions modeling. *arXiv preprint arXiv:1810.12027* (2018).

[27] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning* (2016), PMLR, pp. 1928–1937.

[28] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature 518*, 7540 (2015), 529–533.

[29] Nachum, O., Chow, Y., Dai, B., and Li, L. Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections. *Advances in neural information processing systems 32* (2019).

[30] OpenAI. Spinning up openai. `https://spinningup.openai.com/en/latest/algorithms/ppo.html`, 2018.

[31] R, S. Actor-critic methods. http://incompleteideas.net/book/ebook/node66.html, 2005.

[32] Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438* (2015).

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

[34] Stanford. One hundred year study on artificial intelligence (ai100). https://t.ly/kCEke, 2021.

[35] Sutton, R. S. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin 2*, 4 (1991), 160–163.

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

[37] Sutton, R. S., Mahmood, A. R., and White, M. An emphatic approach to the problem of off-policy temporal-difference learning. *The Journal of Machine Learning Research 17*, 1 (2016), 2603–2631.

[38] Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems 12* (1999).

[39] Sutton, R. S., Precup, D., and Singh, S. Intra-option learning about temporally abstract actions. In *ICML* (1998), vol. 98, pp. 556–564.

[40] Thodoroff, P., Anand, N., Caccia, L., Precup, D., and Pineau, J. Recurrent value functions. *arXiv preprint arXiv:1905.09562* (2019).

[41] Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2012), IEEE, pp. 5026–5033.

[42] Veeriah, V., Zahavy, T., Hessel, M., Xu, Z., Oh, J., Kemaev, I., van Hasselt, H. P., Silver, D., and Singh, S. Discovery of options via meta-learned subgoals. *Advances in Neural Information Processing Systems 34* (2021), 29861–29873.

[43] Velu, A., Vaidyanath, S., and Arumugam, D. Hindsight-dice: Stable credit assignment for deep reinforcement learning. *arXiv preprint arXiv:2307.11897* (2023).

[44] Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature 575*, 7782 (2019), 350–354.

[45] Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224* (2016).

[46] Weng, J., Lin, M., Huang, S., Liu, B., Makoviichuk, D., Makoviychuk, V., Liu, Z., Song, Y., Luo, T., Jiang, Y., Xu, Z., and Yan, S. EnvPool: A highly parallel reinforcement learning environment execution engine. In *Advances in Neural Information Processing Systems* (2022), S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35, Curran Associates, Inc., pp. 22409–22421.

[47] Witten, I. H. An adaptive optimal controller for discrete-time markov environments. *Information and control 34*, 4 (1977), 286–295.

[48] Xu, Z., Modayil, J., van Hasselt, H. P., Barreto, A., Silver, D., and Schaul, T. Natural value approximators: Learning when to trust past estimates. *Advances in Neural Information Processing Systems 30* (2017).

[49] Xu, Z., van Hasselt, H. P., Hessel, M., Oh, J., Singh, S., and Silver, D. Meta-gradient reinforcement learning with an objective discovered online. *Advances in Neural Information Processing Systems 33* (2020), 15254–15264.

[50] Xu, Z., van Hasselt, H. P., and Silver, D. Meta-gradient reinforcement learning. *Advances in neural information processing systems 31* (2018).

[51] Yang, H., Liu, X.-Y., Zhong, S., and Walid, A. Deep reinforcement learning for automated stock trading: An ensemble strategy. In *Proceedings of the First ACM International Conference on AI in Finance* (2020), pp. 1–8.

[52] Yu, C., Liu, J., Nemati, S., and Yin, G. Reinforcement learning in healthcare: A survey. *ACM Computing Surveys (CSUR) 55*, 1 (2021), 1–36.

[53] Yu, T., and Zhu, H. Hyper-parameter optimization: A review of algorithms and applications. *arXiv preprint arXiv:2003.05689* (2020).

[54] Zahavy, T., Xu, Z., Veeriah, V., Hessel, M., Oh, J., van Hasselt, H. P., Silver, D., and Singh, S. A self-tuning actor-critic algorithm. *Advances in neural information processing systems 33* (2020), 20913–20924.

[55] Zheng, Z., Oh, J., Hessel, M., Xu, Z., Kroiss, M., Van Hasselt, H., Silver, D., and Singh, S. What can learned intrinsic rewards capture? In *International Conference on Machine Learning* (2020), PMLR, pp. 11436–11446.

# Appendix A

## A.1 Generalized Beta Advantage Properties

To clarify the properties of Generalized Beta Advantage Estimation (GBAE) under varying $\lambda$ values, we present a demonstration using for the last three time-steps of an episode. This example serves to illustrate that when $\lambda = 1$, GBAE yields the same results as $G_t^\beta - V(s_t)$. Additionally, when $\lambda = 0$, GBAE is the one-step Temporal Difference (TD) advantages $\delta_t = r_t + \gamma v_{t+1} - v_t$. The following analysis highlights the adaptability of GBAE to different learning scenarios, depending on the selected parameter values.

Generalized Beta Advantage Estimation (GBAE)

$$
\begin{aligned}
A_t^{GBAE(\gamma,\lambda)} &= \delta_t + (1-\beta(s_{t+1}))(\gamma\lambda)A_{t+1}^{GBAE} \\
A_T^{GBAE} &= \delta_T + \gamma\lambda A_{T+1}^{GBAE} = r_T - V_T \\
A_{T-1}^{GBAE} &= \delta_{T-1} + \gamma\lambda(1-\beta(s_T))A_T^{GBAE} \\
&= r_{T-1} + \gamma\lambda(1-\beta(s_T))r_T + \gamma V_T - \\
&\quad \gamma\lambda(1-\beta(s_T))V_T - V_{T-1} \\
A_{T-2}^{GBAE} &= \delta_{T-2} + \gamma\lambda(1-\beta(s_{T-1}))A_{T-1}^{GBAE} \\
&= r_{T-2} + (\gamma\lambda)(1-\beta(s_{T-1}))r_{T-1} + \\
&\quad \gamma^2\lambda^2(1-\beta(s_{T-1}))(1-\beta(s_T))r_T + \\
&\quad \gamma V_{T-1} - \gamma\lambda(1-\beta(s_{T-1}))V_{T-1} - \\
&\quad \gamma^2\lambda^2(1-\beta(s_{T-1}))(1-\beta(s_T))V_T - V_{T-2}
\end{aligned}
$$

GBAE when $\lambda = 0$

$$
\begin{aligned}
A_T^{GAE} &= \delta_T = r_T - V_T \\
A_{T-1}^{GAE} &= \delta_{T-1} + \gamma\lambda(1-\beta(s_t))A_T^{GAE} \\
&= r_{T-1} + \gamma V_T - V_{T-1} \\
A_{T-2}^{GAE} &= \delta_{T-2} + \gamma\lambda(1-\beta(s_{t-1}))A_{T-1}^{GAE} \\
&= r_{T-2} + \gamma V_{T-1} - V_{T-2}
\end{aligned}
$$

GBAE when $\lambda = 1$

$$
\begin{aligned}
A_T^{GBAE} &= \delta_T + \gamma\lambda A_{T+1}^{GBAE} = r_T - V_T \\
A_{T-1}^{GBAE} &= \delta_{T-1} + \gamma(1-\beta(s_T))A_T^{GBAE} \\
&= r_{T-1} + \gamma(1-\beta(s_T))r_T + \gamma V_T - \\
&\quad \gamma(1-\beta(s_T))V_T - V_{T-1} \\
&= r_{T-1} + \gamma(1-\beta(s_T))r_T + \\
&\quad \gamma V_T \beta(s_T) - V_{T-1} \\
A_{T-2}^{GBAE} &= \delta_{T-2} + \gamma(1-\beta(s_{T-1}))A_{T-1}^{GBAE} \\
&= r_{T-2} + \gamma V_{T-1} - V_{T-2} + \gamma\lambda(1-\beta(s_{T-1}))r_{T-1} \\
&\quad + \gamma^2\lambda(1-\beta(s_{T-1}))V_T \\
&\quad - \gamma\lambda(1-\beta(s_{T-1}))V_{T-1} \\
&\quad + \gamma^2\lambda^2(1-\beta(s_{T-1}))(1-\beta(s_T))r_T \\
&\quad - \gamma^2\lambda^2(1-\beta(s_{T-1}))(1-\beta(s_T))V_T \\
A_{T-2}^{GBAE} &= r_{T-2} + \gamma(1-\beta(s_{T-1}))r_{T-1} \\
&\quad + \gamma^2(1-\beta(s_{T-1}))(1-\beta(s_T))r_T \\
&\quad + \gamma\beta(s_{T-1})V_{T-1} \\
&\quad + \gamma^2(1-\beta(s_{T-1}))\beta(s_T)V_T \\
&\quad - V_{T-2}
\end{aligned}
$$

Beta Returns with baseline $G_t^\beta - v(s)$

$$
\begin{aligned}
G_T^\beta - V_T &= r_T + \gamma[\beta(s_{T+1})V(s_{T+1}) + (1-\beta(s_{T+1}))G_{T+1}^\beta] - V_T \\
&= r_T - V_T \\
G_{T-1}^\beta - V_{T-1} &= r_{T-1} + \gamma[\beta(s_T)V(s_T) + (1-\beta(s_T))G_T^\beta] - V_{T-1} \\
&= r_{T-1} + \gamma[\beta(s_T)(V_T) + (1-\beta(s_T))r_T] - V_{T-1} \\
&= r_{T-1} + \gamma(1-\beta(s_T))r_T + \gamma\beta(s_T)V_T - V_{T-1} \\
G_{T-2}^\beta - V_{T-2} &= r_{T-2} + \gamma[\beta(s_{T-1})V(s_{T-1}) + (1-\beta(s_{T-1}))G_{T-1}^\beta] - V_{T-2} \\
&= r_{T-2} + \gamma[\beta(s_{T-1})(V_{T-1}) + (1-\beta(s_{T-1}))[r_{T-1} + \\
&\quad \gamma[\beta(s_T)V_T + (1-\beta(s_T))r_T]]] - V_{T-2} \\
G_{T-2}^\beta - V_{T-2} &= r_{T-2} + \gamma(1-\beta(s_{T-1}))r_{T-1} \\
&\quad + \gamma^2(1-\beta(s_{T-1}))(1-\beta(s_T))r_T \\
&\quad + \gamma\beta(s_{T-1})V_{T-1} \\
&\quad + \gamma^2(1-\beta(s_{T-1}))\beta(s_T)V_T \\
&\quad - V_{T-2}
\end{aligned}
$$